HAMLET: A Hierarchical Agent-based Machine Learning Platform

AHMAD ESMAEILI, Department of Computer and Information Technology, Purdue University
JOHN C. GALLAGHER, Department of Electrical Engineering and Computer Science, University of Cincinnati
JOHN A. SPRINGER and ERIC T. MATSON, Department of Computer and Information Technology, Purdue University

9

Hierarchical Multi-agent Systems provide convenient and relevant ways to analyze, model, and simulate complex systems composed of a large number of entities that interact at different levels of abstraction. In this article, we introduce HAMLET (Hierarchical Agent-based Machine LEarning plaTform), a hybrid machine learning platform based on hierarchical multi-agent systems, to facilitate the research and democratization of geographically and/or locally distributed machine learning entities. The proposed system models machine learning solutions as a hypergraph and autonomously sets up a multi-level structure of heterogeneous agents based on their innate capabilities and learned skills. HAMLET aids the design and management of machine learning systems and provides analytical capabilities for research communities to assess the existing and/or new algorithms/datasets through flexible and customizable queries. The proposed hybrid machine learning platform does not assume restrictions on the type of learning algorithms/datasets and is theoretically proven to be sound and complete with polynomial computational requirements. Additionally, it is examined empirically on 120 training and 4 generalized batch testing tasks performed on 24 machine learning algorithms and 9 standard datasets. The provided experimental results not only establish confidence in the platform's consistency and correctness but also demonstrate its testing and analytical capacity.

CCS Concepts: • **Computing methodologies** → **Multi-agent systems**; **Machine learning;**

Additional Key Words and Phrases: Hierarchical multi-agent systems, hybrid machine learning, distributed machine learning, holonic structures, machine learning platform

ACM Reference format:
Ahmad Esmaeili, John C. Gallagher, John A. Springer, and Eric T. Matson. 2022. HAMLET: A Hierarchical Agent-based Machine Learning Platform. *ACM Trans. Auton. Adapt. Syst.* 16, 3–4, Article 9 (July 2022), 46 pages.
https://doi.org/10.1145/3530191

Authors' addresses: A. Esmaeili (corresponding author), J. A. Springer, and E. T. Matson, Department of Computer and Information Technology, Purdue University, N. Grant St., West Lafayette, IN, 47907; emails: {aesmaei, jaspring, ematson}@purdue.edu; J. C. Gallagher, Department of Electrical Engineering and Computer Science, University of Cincinnati, 2600 Clifton Ave., Cincinnati, OH, 45221; email: gallagj9@ucmail.uc.edu.

1556-4665/2022/07-ART9 $15.00
https://doi.org/10.1145/3530191

1 INTRODUCTION

Machine Learning (ML) is a particularly prevalent branch of Artificial Intelligence that is becoming increasingly relevant to real-life applications, including, but not limited to, economics [6], education [24], agriculture [44], drug discovery [68], and medicine [55]. Such a rapid growth, fueled by either the emergence of new data/applications or the challenges in improving the generality of the solutions, demands straightforward and easy access to the state of the art in the field such that both researchers and practitioners would be able to analyze, configure, and integrate machine learning solutions in their own tasks.

Thanks to ubiquitous computing, the past decade has witnessed an explosion in the volume and dimension of data together with the number of machine learning approaches. Knowledge discovery and learning from large and geo-distributed datasets are the targets of the distributed data mining and machine learning approaches in which the concentration is on eschewing over-engineering of machine-learning models by humans and while improving efficacy and scalability by applying both algorithmic innovation and **high-performance computing (HPC)** techniques to distribute workloads across several machines [70]. Additionally, due to the high diversity of the ML problems and solutions contributed by thousands of multi-disciplinary research communities around the world, it is becoming exhausting, if not impossible, for both experts and non-experts to keep track of the state-of-the-art. To overcome this hardship, a growing number of research endeavors and commercial technologies are being devised, such as Auto-sklearn [25], MLR [8], Rapidminer [39], OpenML [69], and Google BigQuery ML [9], to name a few.

This article strives to propose a platform that provides an organizational scheme for storing, training, and testing machine learning algorithms and data in a decentralized format. The suggested platform is based on multi-agent systems and is meant to be open in the sense that it is not limited to be used with a pre-determined set of machine learning components. **Multi-agent systems (MASs)** offer a number of general advantages with respect to computer-supported cooperative working, distributed computation, and resource sharing. Some of these advantages are [72]: (1) decentralized control, (2) robustness, (3) simple extendibility, and (4) sharing of expertise and resources. The decentralized control is, arguably, the most significant feature of MAS that serves to distinguish such systems from distributed or parallel computation approaches. Decentralized control implies that individual agents, within a MAS, operate in an autonomous manner and are, in some sense, self-deterministic. Robustness, in turn, is a feature of the decentralized control where the overall system continues to operate even though some of the agents crash. Decentralized control also supports extendibility in the sense that additional functionality can be added simply by including more agents. The advantages of sharing expertise and resources are self-evident and have been heavily used in our work.

Similar to ML ecosystem, there has been an immense growth in the size and complexity of MASs during the past decades [51]. Although MASs are considered today to be well-suited to analyze, model, and simulate complex systems, in cases where there is a large number of entities interacting at different levels of abstraction, they fail to faithfully represent complex behaviors with multiple granularities. To deal with this problem, hierarchical systems, as an organizational structure, have attracted the attention of MAS researchers. Today, its contributions to many applications, ranging from manufacturing systems [49], transports [11], cooperative work [2] and radio mobile mesh dimensioning [57], are apparent.

The advantages offered by MAS, together with their intrinsic cooperation and coordination abilities, are particularly applicable to machine learning tasks and **knowledge discovery in data (KDD)**, where a considerable collection of tools and techniques are prevalent [17]. The integration of multi-agent technologies and machine learning techniques is often called agent mining

[12] and can be conducted in two directions, i.e., either using machine learning methods to design and train intelligent agents (also called multi-agent learning) or using multi-agent systems to enhance machine learning or data-mining processes (also called agent-driven machine learning). To put it briefly, machine learning and data-mining tasks can benefit from the multi-agent systems by maintaining the autonomy of data sources, facilitating interactive and distributed approaches, providing flexibility in the selection of sources, improving scalability and supporting distributed data sources, supporting the use of strategies in learning and mining processes, and inherently enabling collaborative learning [60, 74].

This article focuses on addressing the ever-increasing challenges of organizing, maintaining, analyzing, and democratizing the use of machine learning resources. More specifically, the proposed model facilitates the organization of machine learning algorithms and datasets in a multi-level similarity-based architecture, enables storing and conducting machine learning tasks at both local and global scales, democratizes the use of machine learning resources through a simple and intuitive query design, facilitates sharing machine learning tasks and results, simplifies the analysis of machine learning resources by providing automated training and testing algorithms, and finally provides researchers and contributors with the freedom and flexibility of applying customized privacy, integrity, and access strategies.

The proposed approach employs the concepts of organizational multi-agent systems in which the agents represent the existing machine learning elements such as algorithms, datasets, and models along with management and reporting units. The hierarchical organization of the agents in the proposed system is dynamically and distributedly built based on the represented capabilities of the agents and during machine learning training and testing tasks. The used capability and skill-based architecture of the agents also provides the above-mentioned flexibility in adding more sophisticated decision-making processes alongside the basic ML tasks. The suggested platform is evaluated comprehensively using both theoretical and empirical approaches. By the theoretical methods, we prove that a HAMLET-based system makes right decisions, reports correct results, and handles resource shortcomings properly. The theoretical approach also accompanies time and space complexity analyses of its algorithms to make sure that its overloads do not exceed its unique features. In a highly dynamic and communicative environment of multi-agents systems, such theoretical analyses provide an additional assurance that the system operates within its expected limits provided that its assumptions are satisfied. The presented empirical results, however, follows two primary objectives: demonstrating how the platform is expected to be used through example queries and their corresponding results, and showing its flexibility, coherence, and ease of use on practical real-world use cases. We have implemented the framework in Python programming language and utilized the machine learning libraries and resources that are commonly used in the community.

The rest of this article is organized as follows: First, we review some of the noteworthy work relevant to this research in Section 2. Then, we present a detailed description of the proposed platform in Section 3. This is followed by Section 4, which assesses the correctness and the performance of the proposed model theoretically. Section 5 highlights the development aspects of the platform and demonstrates its flexibility and capabilities through real-world machine learning tasks. Section 6 concludes the article and provides suggestion for future work.

2 RELATED WORK

There is a considerable number of reports in the literature about the integration of multi-agents and machine learning systems. In this section, we primarily focus on the application of agent-based techniques in different aspects of a machine learning and data-mining life cycle, and we review some of the works that are relatively more related to our research.

One of the earliest works is the research by Kargupta et al. [42]. In this work, the authors have proposed **Parallel Data Mining Agents (PADMA)**, a data-mining system that uses software agents for local data accessing and analysis, together with a web-based interface for interactive data visualization. PADMA was originally proposed for and tested on a text classification task though its was extended later and successfully used in medical applications [41]. Our proposed platform differs from PADMA in multiple ways. For example, in PADMA the mining agents are local to the sites that data resides but in HAMLET, the agents representing data and algorithms are different entities that communicate over a network. However, unlike HAMLET, PADMA's structure is single level where all the agent are connected to and managed by a single facilitator. Such a simplistic structure is prone to scalability issues and limits the application of the platform for small sets of problems and specific tasks. Gorodetsky et al. claimed that the core problem in agent-based machine learning and data mining is not the algorithms themselves—in many cases these are well understood—but instead the most appropriate mechanisms to allow agents to collaborate [32]. They presented a distributed architecture for classification problems and a set of protocols for a multi-agent software tool. Their suggested architecture comprises two main components to handle source-based data and meta-data—along with a set of specialized agents—to handle queries and classification tasks. Dissimilar to HAMLET, their suggested toolkit is not meant to be general purpose covering various data mining and machine learning tasks at the same time. Peng et al. give an interesting comparison between single-agent and multi-agent text classification in terms of a number of criteria including response time, quality of classification, and economic/privacy considerations [53]. Their collaboration model for the multi-agent setting is based on the work presented in Reference [54], according to which, the agents do not initiate document-based cooperation until they are done with their individual classification tasks. As expected, their reported results are in favor of a multi-agent approach. In Reference [67], an abstract agent-based distributed machine learning and data-mining framework is proposed. The work utilizes a meta-level description for each agent to keep track of its learning process and exchange it with the peers. Such meta-information helps the agents reason about their learning process and hence to improve the results. Unlike this work, HAMLET does not make any contributions on improving the quality of the learning algorithms but focuses more on the organization of ML elements and automating the machine learning tasks.

Agent technology has also been employed in meta-data mining, the combination of results of individual learning agents. Perhaps, the earliest most mature agent-based meta-learning systems are: **Java Agent for Meta-learning (JAM)** [63], a collective data mining–based experimental system called BODHI [40], and Papyrus [7]. Basically, all these systems try to combine local knowledge to optimize a global objective. In JAM, a number of learners are simultaneously trained on a number of data subsets, and the results are combined via a meta-learning technique. In BODHI, however, the primary goal is to provide an extensible system that hosts and facilitates the communications between mobile data-mining agents through a three-level hierarchy. In contrast to JAM and BODHI, Papyrus can move not only models but also data from site to site when such a strategy is desired. Papyrus is a specialized system that is designed for clustering while JAM and BODHI are designed for data classification. Apart from their specializations in limited sets of applications, these systems differ from HAMLET in structural aspects as well. For example, BODHI uses a hierarchical structure like HAMLET; however, its structure comprises limited and fixed three levels, whereas HAMLET's architecture does not have any predetermined arrangements. However, the multi-level architecture of BODHI is based on the role of the agents, i.e., whether they are mining agents or facilitators; however, the levels in HAMLET are formed during the lifetime of the systems and based on the learning capabilities of the agents. And last but not least, in most of the similar platforms that tend to perform agent-based machine learning tasks, the structure is configured

and set at the design time based on the objectives of the system. In HAMLET, however, the system starts with basic requirements and sets of protocols and dynamically evolves during its lifetime based on the ML tasks that are conducted on it.

There is very limited work reported on generic approaches. One example is **EMADS (Extendable Multi-Agent Data Mining Framework)** [3]. EMADS has been evaluated using two data-mining scenarios: **Association Rule Mining (ARM)** and Classification. This framework can find the best classifier providing the highest accuracy with respect to a particular dataset. EMADS uses fixed protocols and, since the mining tasks are managed by a single site, called mediator, it is not easily scalable to cover large number of learning agents. Our proposed method, however, not only provides the analytical information about the best learners but also is capable of conducting batch learning tasks, is not limited to classification, and is easily scalable to hold thousands of elements. Liu et al. proposed a hierarchical and parallel model called DRHPDM that utilized a centralized and distributed mining layers for the data sources according to their relevance and a global processing unit to combine the results [46]. In their work, the MAS has aided their approach to realize a flexible cross-platform mining task. The proposed architecture relies on a central GPU unit to determine the relevance of the data sources, forming local mining groups, and processing the final models. Unlike HAMLET, which is capable of performing various machine learning tasks with different configuration parameters, DRHPDM merely focuses on a single mining tasks at a time. Furthermore, it does not clearly specify how the central unit would cope with the use of customized learning algorithms at large scales. In Reference [13], a multi-agent-based clustering framework called MABC is suggested. This framework utilizes four types of agents, namely, user, data, validation, and clustering, to perform the clustering task in two phases: first generating the initial configuration and then improving them through the negotiation among the agents. This work differs from our proposed platform in the sense that it is used only for clustering with predetermined set of algorithms and concentrates more on improving the clustering results than providing analytical information. **ActoDatA (Actor Data Analysis)** [48] is another framework that utilizes an agent-based architecture to provide a set of software tools for distributed data mining and analysis. ActoDataA's objectives differ from our proposed platform, as they are primarily focused on classification tasks and automating different phases of its workflow.

There are numerous other works in the literature that utilize agent technologies and multi-agent systems to improve mining and learning results in specific applications. Some of the noteworthy researches in this category are: Reference [10], which introduces a lightweight mobile agent platform called **JavaScript Agent Machine (JAM)** that couples machine learning with multi-agent systems with an application in earthquake and disaster monitoring; Reference [73], which proposes an agent-based architecture for cyber-physical manufacturing systems; Reference [66], which uses multi-agent coordination in credit risk mining process; Reference [30], which leverages agent technologies in intrusion detection systems, and Reference [31], which suggests a three-level agent-based clustering framework for satellite networks. As these researches are not directly related to objectives of our article, we do not delve into them further.

3 PROPOSED APPROACH

3.1 Problem Formalization

The principal components of any machine learning system are the implemented algorithms, the stored datasets, and the visualization and/or report generators. In the proposed platform, we concentrate on the first two parts, though it can be easily extended to support as many additional sections as required. Let us assume that each algorithm or dataset entity in such a system is represented by a node. These nodes are connected by training/testing queries initiated by

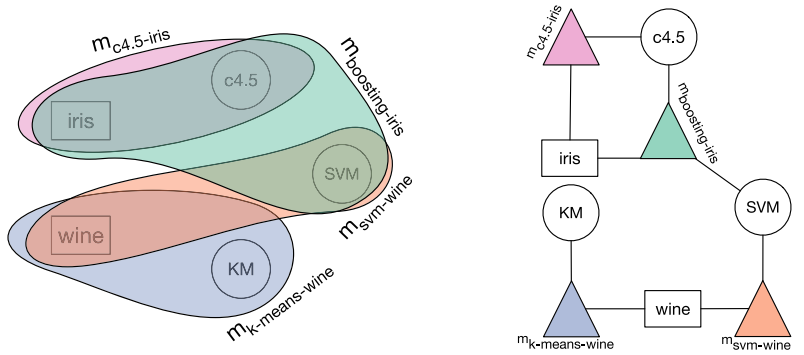


Fig. 1. An example machine learning system as a hypergraph.

end-users. Since the models might involve more than two entities, we model the entire system as a hypergraph. The nodes of the hypergraph are the data/algorithm entities, and the hyperedges are the models that are built during training/testing procedures. Formally, we denote the system hypergraph by tuple $S\langle X, M\rangle$ where M is the set of built models and X is the set of all available data and algorithms, that is, $X = \{a_1, a_2, \ldots, a_N, d_1, d_2, \ldots, d_L\}$ where a_i and d_j represent an specific algorithm and data, respectively. Figure 1(a) depicts an example system composed of three algorithms: $\{svm, k\text{-}means(\text{KM}), c4.5\}$; and two datasets: $\{iris, wine\}$ that are used in three models $m_{svm\text{-}wine}, m_{c4.5\text{-}iris}$, and $m_{k\text{-}means\text{-}wine}$. In other words, $X = \{svm, k\text{-}means(\text{KM}), c4.5, iris, wine\}$ and $M = \{m_{svm\text{-}wine}, m_{c4.5\text{-}iris}\}$. In this research, we use the multi-modal representation of the hypergraph in which a hyperedge is translated into a new type of node that is connected to all the entities it contains. Figure 1(b) depicts the multi-modal representation of the aforementioned example of a machine learning system.

The following are the definitions and the assumptions that are used to present the proposed model:

- *Algorithm* refers to any machine learning method, e.g., clustering or classification, in its abstract form without taking any specific data into account. An algorithm is denoted by tuple $a_i\langle name, type, P_{a_i}\rangle$, where $P_{a_i} = \{(p_i, v_i)\}$ denotes the set of parameters that is used for the configuration of the algorithm, and $type \in \{classification, clustering, regression, \ldots\}$ specifies the category that this algorithm can belong to. The *type* term has an organizational purpose and does not affect the functionality of the algorithm.
- *Dataset* refers to all pre-stored sets of instances used for machine learning tasks. Similar to an algorithm, a dataset is represented by tuple $d_i\langle name, type, P_{d_i}\rangle$, where P_{d_i} specifies a set of access parameters in the dataset and *type* denotes the chain categories that the data belongs to, as described above.
- *Model* refers to the working implementation of a machine learning method based on the above-mentioned algorithms and datasets. For the sake of simplicity, we concentrate only on the models that implement a single algorithm on a single dataset in this article. Such a model is denoted by tuple $m_i\langle name, a, d, P_{m_i}\rangle$, where a and d are, respectively, the algorithm and dataset on which the model is based; and $P_{m_i} = \{(p_i, v_i)\}$ specifies the model-specific configuration parameters.
- *Query* refers to the machine learning requests that an end-user sends to the system. Throughout this article, we assume that any query, training, or testing is specified by

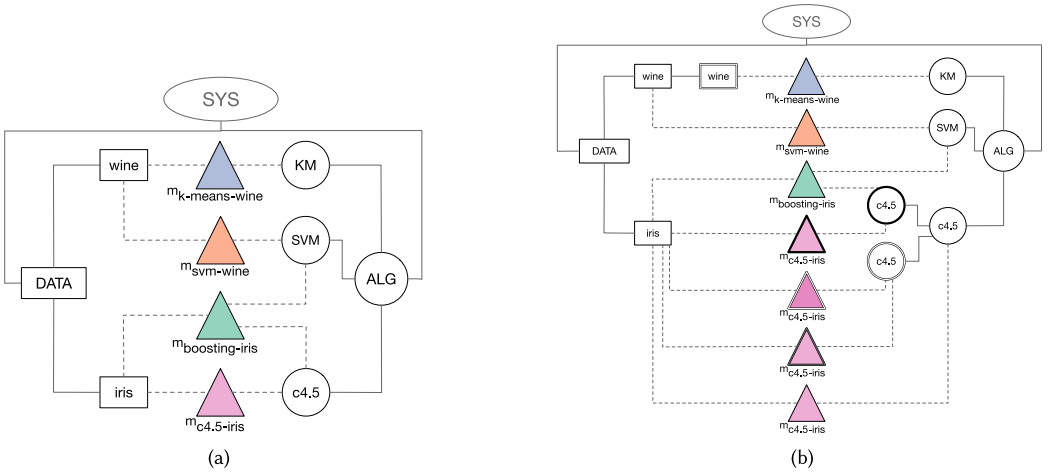


Fig. 2. The hierarchical representation of (a) the machine learning hypergraph example in Figure 1, and (b) its extended version.

tuple $q_i \langle id, \Lambda_i, \Delta_i, O_i \rangle$, where id uniquely identifies the query; $\Lambda_i = \{(a_j, P_{a_j})\}$ and $\Delta_i = \{(d_k, P_{d_k})\}$, respectively, denote the sets of algorithms and data that are used; and finally, O specifies the configuration of the output that the end-user is interested in. For instance, the tuple

$$< 00, \{(svm, \{(kernel, rbf)\}), (c4.5, \{\})\}, \{(iris, \{\})\}, \\ \{type\text{=}test,\ format\text{=}plot, measures = \{accuracy\}\} >$$

specifies a query that tests algorithms *svm*, with *rbf* kernel, and *c*4.5, with default parameters, on *iris* dataset, with default parameters, and plots the accuracy as the result. It is clear that the number of different machine learning operations in a single query q_i, by this definition, is $|\Lambda_i| \times |\Delta_i|$, where $|...|$ denotes the set cardinality.

3.2 Multi-agent Modeling

The holistic view of the proposed model has a hierarchical structure of the agents that are specialized in managing data or processing a machine learning task. We use the aforementioned multi-modal representation of the machine learning hypergraph to express the structure of the multi-agent hierarchy, in terms of its components and the communication links. More specifically, each modal of the graph constructs a sub-tree that holds all the similar data/algorithm agents. For instance, Figure 2(a) depicts the hierarchical modeling of the example in Figure 1. Please note that we have used two different styles (solid and dashed lines) to draw the links between the structural components in this figure. This is for the sake of clarity in exhibiting the tree-like relationship among the components of the **algorithm (ALG)** and **data (DATA)** sub-structures. Figure 2(b) tries to show an extended version of Figure 2(a) system to reflect its scalability for real-world scenarios.

The high flexibility and the multi-level structure of the proposed method requires an adaptable multi-agent organization model. In this article, we use a self-similar and hierarchical structure called a holarchy to model the entire proposed machine learning platform. The next section provides a short introduction to holon and holarchies together with the details on how we have adopted them in our solution.

3.2.1 The Holonic Machine Learning Model.

Holonic Multi-agent Systems. The term holon was introduced by Arthur Koestler [43] to explain the self-similar structure of biological and social systems. He made two key observations: these systems evolve and grow to satisfy increasingly complex and changing needs by creating stable "intermediate" forms that are self-reliant and more capable than the initial systems, and it is generally difficult to distinguish between "wholes" and "parts" in all living and organizational systems. Put another way, almost every distinguishable element is simultaneously a whole (an essentially autonomous body) and a part (an integrated section of a larger, more capable body). The concepts of holonic systems have been successfully adopted and used in the design of organizational multi-agent systems. For instance, Reference [16] introduces ASPECS, a step-by-step software process for modeling and engineering complex systems at different levels of details using a holonic organizational meta-model. GORMAS [5], however, provides analytical and design methodologies for open virtual organizations, including holonic agent-based systems. In multi-agent systems, the vision of holons is much closer to that of recursive or composed agents. A holon constitutes a way to gather local and global, individual, and collective points of view. Therefore, a holon is a self-similar structure composed of other holons as sub-structures. This hierarchical structure composed of holons is called a holarchy. Depending on the level of observation, a holon can be seen either as an autonomous atomic entity or as an organization of holons. In other words, a holon is a whole-part construct that is not only composed of other holons but it is, at the same time, a component of a higher-level holon.

In a holonic multi-agent system, we can distinguish between two main types of holons, namely, head and body holons. All of the holons, which are members of another holon (called super-holon), are considered to be body holons. These holons can be either atomic or composite and are performing the tasks that have been delegated to them. However, head holons act as the representatives of the holons they are members of. In other words, holons are observable to the outside world by means of these representatives. As the representatives, head holons manage the holons' communication with the outside of the holon and coordinates the body holons in pursuit of the goals of the holon. Inside a holon, the force that keeps the heads and bodies is their commitments to the goal of the holon. It is worth noting that in this commitment, the relationships among the agents and holons are formed at runtime, contrary to classical methods such as object-oriented programming in which they are expressed at the code level. More formally, according to Reference [26], for a MAS with the set A of agents at time t, the set H of all holons is defined recursively as follows:

- Every instantiated agent can be considered as an atomic holon.
- $h = (Head, Sub\text{-}holons, C) \in H$, where $Sub\text{-}holons \in H \setminus \emptyset$ is the set of holons that participate in h; $Head \subseteq Sub\text{-}holons$ is a non-empty set of holons that has the aforementioned managerial responsibilities; and $C \subseteq Commitments$ defines the relationship inside the holon and is agreed on by all holons $h' \in Sub\text{-}holons$ at the time of joining the holon h. These commitments keep the members inside the holon.

According to the definitions above, a holon h is observed by the outside world of h like any other agent in A. Only at a closer inspection, it may turn out that h is constructed from (or represents) a set of agents. Like traditional agents, any holon has a unique identification. This facilitates communication among the holons by just sending messages to their addresses. Handling these messages is one of the responsibilities of the head holon(s). Given the holon $h = (Head, \{h_1, \ldots, h_n\}, C)$, we call $h_1, \ldots, h_n$ the sub-holons of h, and h the super-holon of $h_1, \ldots, h_n$. The set $Body = \{h_1, \ldots, h_n\} \setminus Head$ (the complement of Head) is the set of sub-holons that are not allowed to represent holon h. Holons are allowed to engage in several different super-holons with noncontradictory goals.

The Adoption of the Holonic Concepts. Based on the formalization of the machine learning problem that we presented in Section 3.1, we define a new holon for each of the structural components of the machine learning hierarchy. Generally, these holons can be categorized as follows:

- **SysH**, as the system holon, acts as the representative of the entire machine learning system to the outside world.
- **AbsH**, as the set of abstract holons, acts as the container for all holons of the same functionality. These types of composite holons do not directly perform any machine learning task or manage a dataset, and hence are not connected to any **model holon (ModH)**. Instead, they play a managerial role in handling the queries and the results. There are at least two pre-defined abstract holons in the proposed platform: *abstract data* and *abstract algorithm* holons that while are the sub-holons of the *SysH* act as the ultimate parent of all existing data and algorithm holons in the system, respectively. These holons are specified by "DATA" and "ALG" in Figure 2(b). The *type* parameter of an algorithm or data tuples, can be used to define a hierarchy of the abstract holons. That is, the system might have other abstract holons for categorization and managerial purposes, such as an abstract holon to hold all the decision tree algorithms or to manage all the categorical data.
- **DatH** refers to the set of non-abstract holons that hold and manage datasets. A data holon is atomic and is part of an AbsH. Moreover, when a dataset is not used in the training of any machine learning model, its corresponding DatH is not connected to any model holon, ModH.
- **AlgH** refers to the set of non-abstract holons that contain the implementation and configuration details of a data-mining algorithm. Similar to a DatH, an AlgH is atomic and might or might not maintain links with model holons.
- **ModH**, as the set of model holons, corresponds to a realization of a machine learning algorithm used on a specific dataset. A ModH is an atomic holon constructed by an AlgH and is the sub-holon of a DataH in addition to its creator AlgH. It is also assumed that a ModH does not interact with the other model holons.

To show the hierarchical relationship between these holons, let us assume that ${}^{U}_{t}H^{l}_{i}$ denotes the ith holon of type $t \in T = \{s, a, d, m\}$ at level l of the holarchy, where s, a, d, and m stand for *system, algorithm, data*, and *model*, respectively; and U is the set containing the indexes of its super-holons. Numbering the levels from top of the holarchy starting from 0, we have:

$$SysH = {}^{\emptyset}_{s}H^{0}_{0}, \tag{1}$$

$$AbsH = \left\{ {}^{\{u\}}_{t}H^{l}_{i} : t \in \{a, d\} \ \wedge \ l > 0 \ \wedge \ {}^{\{u'\}}_{t}H^{l-1}_{u} \in (AbsH \cup SysH) \right\}, \tag{2}$$

$$DatH = \left\{ {}^{\{u\}}_{d}H^{l}_{i} : l > 1 \ \wedge \ {}^{\{u'\}}_{d}H^{l-1}_{u} \in (DatH \cup AbsH) \right\}, \tag{3}$$

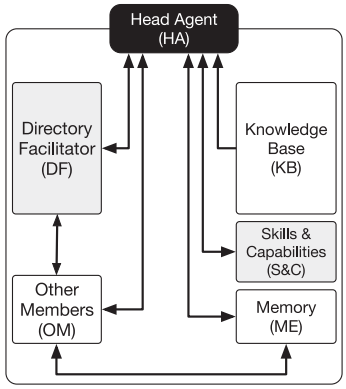
$$AlgH = \left\{ {}^{\{u\}}_{a}H^{l}_{i} : l > 1 \ \wedge \ {}^{\{u'\}}_{a}H^{l-1}_{u} \in (AlgH \cup AbsH) \right\}, \tag{4}$$

$$ModH = \left\{ {}^{\{u,u'\}}_{m}H^{l}_{i} : l > 2 \ \wedge \ {}^{\{x\}}_{a}H^{l-1}_{u} \in AlgH \ \wedge \ {}^{\{z\}}_{d}H^{l-1}_{u'} \in DatH \right\}, \tag{5}$$

and hierarchical relationship between the holons are defined as follows:

$${}^{\emptyset}_{s}H^{0}_{0} = \left\{ {}^{\{0\}}_{a}H^{1}_{1}, {}^{\{0\}}_{d}H^{1}_{1} \right\}, \tag{6}$$

$${}^{U}_{t}H^{l}_{i} = \left\{ {}^{\{i\}}_{t}H^{l+1}_{j} : l \geq 1 \ \wedge \ t \in \{a, d\} \right\} \ \cup \ \left\{ {}^{V}_{m}H^{k}_{j} : i \in V \right\}. \tag{7}$$



(a)

	HA	DF	OM	KB	S&C	ME
SysH	✓	✓	✓	✗	✗	✓
AbsH	✓	✓	✗	✗	✓	✓
AlgH	✓	✓	✓	✓	✓	✓
DatH	✓	✓	✓	✓	✓	✓
ModH	✓	✓	✗	✓	✓	✓

(b)

Fig. 3. The general architecture of a holon.

Please note that these statements merely define the holonic relationships and do not identify the complete set of members that a holon might have.

Apart from its members, a holon comprises several other parts that enable it to process and answer the queries. The general internal architecture that is commonly used by the above-defined holons is shown in Figure 3(a). In this figure, the arrows show the direction of the information provided between the components; the ***Head Agent*** **(HA)**, administers the inter- and intra- holon interactions; the ***Knowledge Base*** **(KB)** stores the data or the details of the algorithm; the ***Directory Facilitator*** **(DF)** maintains the information of and accesses to all of the super-holons and members, including sub-holons and the other members; the ***Memory*** **(ME)** component stores the history of the results and useful management logs; the ***Skills & Capabilities*** **(S&C)** element refers to the abilities that distinguish a specific holon from the similar ones; and finally, the ***Other Members*** **(OM)** part contains the utility agents that are employed by the head to handle the internal tasks, such as query agents, result agents, and so on. It should be noted that the holarchical relationships are maintained using DF, and there is no designated component to hold the sub-holons. This is for the sake of keeping the holon as small as possible and to make it easily distributed over multiple devices. Furthermore, we assume that the identity and category information of the holon is handled by its HA. Depending on the type of the holon, the suggested architecture might miss some parts. Figure 3(b) provides the included components of each holon type in the proposed platform.

As it was mentioned above, the skills and capabilities component distinguishes a holon from its counterparts. In fact, this entity plays a canonical part in the mining operations of the holon and specifying its position inside of the holarchy. In the proposed platform, these terms are defined as follows:

- **Capability**, denoted by C, refers to the innate machine learning ability of a holon, considering itself and all of its members. We use the configuration parameters of the data/algorithm that holon represents to define its set of capabilities. In case of atomic holons, $C_{{}^{U}_{t}H^{l}_{i}} = P_{t_i}$ where P_{t_i} indicates the configuration parameters of the corresponding algorithm/data/model entity. For composite abstract holons, however, the capability is defined as the parametric sum (discussed later in Definition 3.5) of the capabilities of all of its non-model sub-holons.
- **Skill** refers to the set of the specific expertise of the holon. Unlike capabilities that exist intrinsically from the birth of the holon, skills are acquired as soon as the holon is involved

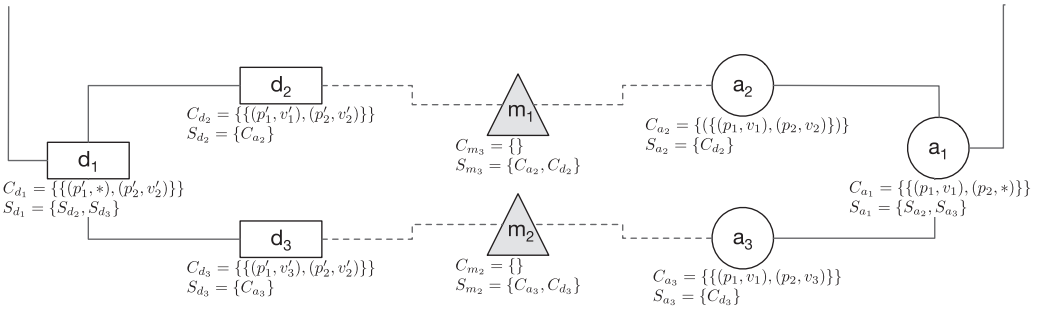


Fig. 4. Capabilities and skills in a simple holarchy example.

in a practical machine learning operation, i.e., the time a child of *ModH* type is spawned. That being said, we update the skill set of *ModH* holons and the other type of holons differently. Formally, if the skill set of the atomic holon ${}^{U}_{t}H^{l}_{i}$ is denoted by $S_{{}^{U}_{t}H^{l}_{i}}$, then we have:

$$S_{{}^{U}_{m}H^{l}_{i}} = \bigcup_{u' \in U} C_{{}^{Y}_{t}H^{l'}_{u'}}; \quad t \in \{a, d\}, \tag{8}$$

$$S_{{}^{U}_{t}H^{l}_{i}} = \bigcup_{Y} \left\{ \{C_{{}^{Y}_{m}H^{\lambda}_{j}}; i \in Y\} \setminus C_{{}^{U}_{t}H^{l}_{i}} \right\}. \tag{9}$$

Additionally, for the case of an abstract data/algorithm holon, the skill set is the union of the skills of its sub-holons. That is:

$$S_{{}^{u}_{t}H^{l}_{i}} = \bigcup_{u'} S_{{}^{(i)}_{t}H^{l+1}_{u'}}. \tag{10}$$

By way of explanation, the skill set of the model holon is the union of the capabilities of its super-holons, and the skill set of the atomic data/algorithm holons is the union of the skills of its model sub-holons without its own capability set. Finally, the skill set of a composite abstract holon comprises the combination of subordinate skills. The toy example of Figure 4 demonstrates how the capabilities and skills are set in a partial holarchy.

It should be emphasized that these terms are defined with respect to machine learning tasks and does not take the intrinsic abilities of the other members of the holon, such as processing the queries, generating reports, and so on.

The Construction of the Holarchy. One of the key steps in utilizing a holonic multi-agent model is the initial construction of the hierarchy. The pattern of the holon arrangement in the holarchy plays a critical role in the performance of the system and its dynamic adaptation to the changes of the environment [19–21]. Although a holonic structure can be designed and hand-arranged by an expert, there are numerous research endeavors in the literature that have concentrated on the automatic organization of holonic multi-agent systems—socially based method proposed in References [19, 21], the RIO [36]-based approach in Reference [37], and the Petri net-based model reported in Reference [15], to name a few. In this article, The holarchy is initially composed of a *SysH* and two *AbsH*s, namely, ALG and DATA, to accommodate all algorithms and data of the system, respectively. Then it dynamically continues growing as new machine learning queries are sent to the system. The detailed algorithm makes use of a few new operators and symbols defined as follows:

Definition 3.1 (Parametric Set). A parametric set P is a set of parameter-value ordered pairs (p_i, v_i) such that $\exists (p_j, v_j) \in P, p_i = p_j \implies v_i = v_j$. In other words, there is only one pair for any parameter p_i in the set.

Definition 3.2 (Parametric General Symbol). A parametric general symbol, denoted by $\star$, is a placeholder for all the available values of a parameter. For instance, the ordered pair $(learning\_rate, \star)$ implies the set of all available values for parameter $learning\_rate$. Throughout this article, we call a pair general if it the symbol appears as its value; and similarly, we call a set general if it contains at least one general pair.

Definition 3.3 (Parametric Congruence). Two ordered pairs (p, v) and (p', v') are called parametric congruent, written as $(p, v) \overset{\star}{\cong} (p', v')$, if and only if $p = p'$. Similarly, congruence for two parameter sets P and P' is defined as follows:

$$P \overset{\star}{\cong} P' \iff |P| = |P'| \ \wedge \ \forall (p, v) \in P \ \exists (p', v') \in P' : (p, v) \overset{\star}{\cong} (p', v'). \tag{11}$$

Definition 3.4 (Parametric Inequality). The ordered pair (p, v) is parametrically less than or equal to $(p', v\prime)$, denoted by $(p, v) \overset{\star}{\leq} (p', v\prime)$, if and only if they are congruent and $v = v' \vee v = \star \vee v' = \star$. Similarly, for two sets P and P', we have:

$$P \overset{\star}{\leq} P' \iff \forall (p, v) \in P \ \exists (p', v') \in P' : (p, v) \overset{\star}{\leq} (p', v'). \tag{12}$$

Please note that P and $P\prime$ sets do not necessarily need to be parametric congruent. For instance, $\emptyset \overset{\star}{\leq} \{(p_1, v_1), (p_2, v_2)\}$; $\{(p_1, v_1)\} \overset{\star}{\leq} \{(p_1, \star), (p_2, v_2)\}$; $\{(p_1, v_1), (p_2, \star)\} \overset{\star}{\leq} \{(p_1, \star), (p_2, v_2)\}$ all yield true, whereas $\{(p_1, v_1), (p_2, v_2)\} \overset{\star}{\leq} \{(p_1, \star), (p_2, v_3)\}$ results in false.

Definition 3.5 (Parametric Sum). Parametric sum, denoted by $\overset{\star}{+}$, is a binary operator defined on parametric congruent pairs or sets as follows:

$$(p, v) \overset{\star}{+} (p, v') = \begin{cases} (p, v) & \text{if } v = v' \\ (p, \star) & \text{if } v \neq v', \end{cases} \tag{13}$$

$$P \overset{\star}{+} P' = \bigcup_{\substack{(p,v) \in P \\ (p,v') \in P'}} \{(p, v) \overset{\star}{+} (p, v')\}. \tag{14}$$

Furthermore, parametric sum has the additive identity property over sets, i.e $\emptyset \overset{\star}{+} P' = P'$.

Definition 3.6 (Parametric Similarity Ratio). Parametric similarity ratio, denoted by $\overset{\star}{\sim}$, is a bivariate function that quantifies the similarity between two parametric congruent pairs or sets, in terms of their common values. The range of $\overset{\star}{\sim}$ lies in $(0, 1]$ and is predefined for all parameter values. In this article, we use the following definition:

$$\overset{\star}{\sim}((p, v), (p, v')) = \begin{cases} 1 & \text{if } v = v' \\ \alpha & \text{if } v \neq v' \wedge (v = \star \veebar \ v' = \star) \\ \beta & \text{if } v \neq v' \wedge (v \neq \star \wedge \ v' \neq \star), \end{cases} \tag{15}$$

such that $0 < \beta < \alpha < 1$. The parametric difference of two parameter sets P and P' is defined as follows:

$$\overset{\star}{\sim}(P, P') = \frac{\sum_{v=v'} \overset{\star}{\sim}((p, v), (p, v')) + \prod_{v \neq v'} \overset{\star}{\sim}((p, v), (p, v'))}{|P|}, \tag{16}$$

where $(p, v) \in P$; $(p, v') \in P'$; and $|\ldots|$ denotes the set cardinality. That being said, the possible values of the parametric similarity score of two sets lie in range $(\frac{\beta^{|P|}}{|P|}, 1]$. In this article, we have

set $\alpha = 0.5$ and $\beta = 0.1$. It can be easily shown that the operators explained in Definitions 3.3, 3.5, and 3.6 all have commutative property.

As it was stated before, the construction of the holarchy begins with the initial SYS = {DATA, ALG} holons. For now, let us assume that the user query is properly processed by a utility member of SYS holon and is sent to its both sub-holons. The very first thing that the abstract ALG holon checks, upon receiving the query from the SYS, is the name of the algorithm to direct the request to a proper path down the holarchy. For this purpose, it initiates a bidding process based on **Contract Net Protocol (CNP)** [62] and asks its sub-holons for their proposals. The proposal of its immediate sub-holon $h = {}_{a}^{\{1\}}H_i^2$ is basically the result of $\overset{\star}{\sim}((name, a^q_{name}), (name, h_{name}))$, where a^q_{name} and h_{name} are the names of the new algorithm requested to be added by the query and the name of the holon h, respectively. Having received the proposals, the ALG holon chooses the sub-holon with proposal value 1—there will be only one such proposal—to forward the query to. If there is no such proposal, then ALG spawns a new holon to represent the new algorithm's specifications. Algorithm 1 presents the details of the process. The names of the variables and functions are chosen to be self-explanatory, and we have left comments wherever further explanations are needed.

ALGORITHM 1: The new algorithm initiation function.

```
   Input: a_i ⟨name_{a_i}, type_{a_i}, P_{a_i}⟩ // an algorithm
1  best ← myself;                    // myself refers to the holon that calls this function
2  foreach {}_a^{1}H_i^2 ∈ {}_a^{0}H_i^1 do
3      proposal ← CFP({}_a^{1}H_i^2, (name, name_{a_i}));          // Call For Proposal
4      if proposal = 1 then
5          best ← {}_a^{1}H_i^2;
6          break;
7      end
8  end
9  Ask(best, "ADD", a_i);          // ask the ''best'' holon to add algorithm a_i as new holon
```

When an algorithm holon is asked to add a new holon (line 9 of Algorithm 1), it runs the AddORTrainFirstPass function, in which a slightly different set of steps is followed. The first difference pertains to the way proposals are generated and chosen. Unlike before, the holons use their capability sets to calculate their similarity scores, and the one with the greater value is chosen. Furthermore, if the chosen holon is atomic, then a new super-holon is created to contain both the chosen and the new holon. Otherwise, a new sub-holon is created for the algorithm under the current holon, or the new algorithm info will be passed down recursively until it is handled properly. The details of the process is presented in Algorithm 2. Because the addition algorithm shares many steps with the first run of the training phase detailed in the following section, we combined them into a single algorithm and used comments and red color to highlight the parts that are relevant to a certain job. In this algorithm, the arguments of function CreateNewHolon (lines 3, 10, and 27) are the name of the holon, the reference to its super-holon, the set of its capabilities, and finally the set of its skills, respectively. Additionally, there are a few utility functions that are called regularly in this algorithm. Function Ask in lines 9, 12, and 31, sends a request to the holon, specified in the first argument, to run an operation, specified in the second argument, based on the information that are provided in the remaining arguments. Upon receiving such a request, the holons call appropriate functions and answer the request accordingly. Functions JoinHolon and UpdateCapability, however, are used by the holon to properly move from one super-holon to another, and update its capability lists, respectively. The details of these two functions are presented in Appendix B,

Algorithm 5. In this algorithm, function Inform is used to let a holon know about some facts. In the case of the presented functions, it is used to inform the super-holon about the updates in capabilities so the super-holon makes the necessary updates accordingly. In all of these algorithms, it is assumed that holon ${}^{\{u\}}_{a}H^{l>0}_{i}$ is the one that calls and uses the functions. Appendix C.1 uses an example to describe the process in detail.

3.2.2 Training and Testing. In the previous section, we discussed how the holarchy is built as new algorithms or data are added to the system. The important system component that was not taken into consideration in the aforementioned process is the model holon. As we have mentioned earlier, the model holons are to represent a practical realization of applying an algorithm on a dataset, therefore, there was no need for their creation, as we merely added the definitions of algorithms to the holarchy. In this section, we present the details of the holarchical growth and alternations that happen when we try to train or test an algorithm on specific datasets.

Training. By training, we mean creating a machine learning model that is tuned to answer queries about a specific dataset. In the proposed platform, when the system is asked to train a particular algorithm on a specified dataset, one of the following cases happens:

- The holarchy contains no holons that represent the data/algorithm or both of them. In this case, if enough information is provided about the missing component(s), then they are added to the holarchy together with the requested model.
- The holarchy contains both the data and the algorithm but not the model. In this case, a model holon is created and properly linked to the data and algorithm holons.
- The holarchy contains the model, i.e., it has been trained before, with exactly the same configurations. Here, the system might inform the user about the duplication and provide some already available information.

In the remainder of this section, we assume that each training query contains only one algorithm-data pair information, and whenever there is a need to grow the holarchy, all the information about the data and the algorithm is available. Moreover, we skip the details of the interactions with the user in duplication cases.

To deal with the first two cases of the aforementioned list at once, we follow a procedure very similar to the algorithms of inserting a new component, presented in the previous section. Strictly speaking, two passes will be carried out in the holarchy to train the model. In the first pass, which is presented in detail in Algorithm 2, the holarchy is searched to locate the holons representing the data/algorithm, and in case either or none of the data or algorithm is available, the missing component(s) are added to the holarchy. The auxiliary algorithms that are used to implement the training process are presented in Appendix B. When the holon representing the algorithm is found/created, it is asked to spawn an empty model holon (lines 4, 7, 13, and 29 of the algorithm). Function SpawnModel() creates the model holon properly, stores its address for the current query in the memory, and initiates a recursive address updating procedure in all of the super-holons in the path towards SYS. These addresses are used by all holons in the second phase of the training algorithm to send the commands directly to the appropriate destination without searching for it again. Please recall that the model holons are only created by the algorithm holons, i.e., AlgH, and in the case of DatH holons, the destination will be the atomic data holon that will provide the required data for the query. The details of spawn and address update functions are presented in Appendix B, Algorithm 6.

The second phase of the training procedure begins as soon as the SysH holon is informed about the addresses of the training query from both ALG and DATA sub-holons. In the second pass, SysH asks the ALG and DATA holons to initiate training, being provided the address of the

ALGORITHM 2: The process of adding a new algorithm or running the first pass of training phase.

Input: $a_i \langle name_{a_i}, type_{a_i}, P_{a_i} \rangle$ // an algorithm
Function AddORTrainFirstPass(a_i)
  // myself refers to the holon that calls this function
  if *myself.LEVEL* = 1 **then** // I am ALG holon
    new_holon ← CreateNewHolon($name_{a_i}$, myself, P_{a_i}, {});
    Ask(new_holon, "SPAWN", "MODEL"); // only in AlgHs during training
  else if *AmIAtomic()* **then**
    if $\boxed{\star}(C_{myself}, P_{a_i}) = 1$ **then** // The algorithm already exists
      SpawnModel(); // only in AlgHs during training
    else
      new_super ← Ask(myself.SUPER, "CREATE-HOLON", myself.C, myself.S);
      new_holon ← CreateNewHolon($name_{a_i}, \emptyset, P_{a_i}$, {});
      JoinHolon(new_super);
      Ask(new_holon, "JOIN", new_super);
      Ask(new_holon, "SPAWN", "MODEL"); // only in AlgHs during training
    end
  else
    best ← myself;
    $\varsigma_{\max} \leftarrow \boxed{\star}(C_{myself}, P_{a_i})$;
    foreach ${}^{\{i\}}_{a}H^{l+1}_{j} \in {}^{\{u\}}_{a}H^{l}_{i}$ **do**
      proposal ← CFP(${}^{\{i\}}_{a}H^{l+1}_{j}, P_{a_i}$); // Call For Proposal
      if *proposal* > $\varsigma_{\max}$ **then**
        $\varsigma_{\max}$ ← proposal;
        best ← ${}^{\{i\}}_{a}H^{l+1}_{j}$;
        break;
      end
    end
    if *best = myself* **then**
      new_holon ← CreateNewHolon(myself.NAME, myself, P_{a_i}, {});
      UpdateCapability(P_{a_i});
      Ask(new_holon, "SPAWN", "MODEL"); // only in AlgHs during training
    else
      Ask(best, "ADD", a_i); // only in Add procedure
      Ask(best, "TRAIN FIRST PASS", a_i); // only in AlgHs during training
    end
  end
End Function

corresponding model/data holon (companion). Receiving this request, each holon forwards the request to the address that it has stored in its memory in the previous pass. This will continue until the request reaches the proper destination, i.e., the newly created model holon or the atomic data holon. Upon receiving the request, the data holon is configured to provide access to the specified model holon when needed. However, the model holon, as soon as it receives the training command, communicates with the data holon through the address that has been provided and then starts to train its inherited algorithm on the provided data. Two points should be noted. First, the model

holon joins the data holon and updates its capabilities only after it successfully communicated the data holon and granted access to the data. Second, the skills of the model holon and all of its super-holons are updated after the training procedure finishes successfully, i.e., the algorithm is successfully trained on the data. This update can happen on the way the training results are sent back to the SysH holon. The details of the second pass are presented in Algorithm 3. Although it is out of the scope of this article, it is worth remarking that the two-pass training process facilitates control mechanisms and integrity checks, especially when the data/algorithms are provided by third parties. Furthermore, the model holons, being the shared member of an algorithm and a data holon, help the system properly keep track of and handle the changes potentially made in the definition or access of algorithm/data holons in the entire system.

ALGORITHM 3: The second pass of the training process.

```
1 Function TrainSecondPass(query-id, target-address, companion-address)
      // myself refers to the holon that calls this function
2     if target-address = myself.ADDRESS then
3         StartTraining(companion-address);
4     else
5         address ← GetAddress(query-id);
6         Ask(address, "TRAIN SECOND PASS", query-id, companion-address);
7     end
8 End Function
```

As it has been mentioned before, the proposed platform is designed in such a way that no operation blocks the flows of the past or future ones. In other words, while a specific agent/holon is busy processing a query, the other parts are open to accepting new requests without waiting for the previous results. This behavior is largely managed by the communication of the holons as described before, together with the local updates that each holon might make as needed. For instance, the first pass of a new training query that is being processed just before the second pass of another query begins can easily invalidate the previous address updates due to the potential structural changes it may cause. To overcome this problem, whenever a new data/algorithm holon is created, the holons in the vicinity of the change update their addresses accordingly. Appendix C.2 depicts this process in more details.

Testing. We define testing identically with how it is contemporarily used in the machine learning/data science communities, which is a metric-based assessment of the efficacy of a trained system against particular pre-defined and ground-truthed datasets. The prerequisite of exerting a testing operation is defined by each atomic holon. Without loss of generality, the following testing method assumes that the test data is already in the holarchy and its information is explicitly provided. This needs a process very similar to the one we used in the first pass of the training algorithm (Algorithm 2) to insert and/or retrieve the address of the test dataset.

The testing process launches by the SYS holon passing the testing information and criteria to the ALG holon. As we would like to process and perform the operations in a batch, we allow the use of the parametric general symbol (defined in Definition 3.2) in the query. Receiving the request, each holon compares the criteria, consisting of the names and configuration parameters of the data and algorithms, with its own capabilities and skills, based on the definition of parametric inequality (Definition 3.4). Formally, let (a_j, P_{a_j}) and (d_j, P_{d_j}) be the specifications of the testing algorithms and data, respectively. As soon as holon ${}^{\{u\}}_{a}H^{l}_{i}$ receives a testing request from its super-holon, it

checks if the following statement is true:

$$\left((name, a_j) \overset{\star}{\leq} \left(name, name_{\underset{a}{\{u\}}H_i^l}\right)\right) \wedge \left(P_{a_j} \overset{\star}{\leq} C_{\underset{a}{\{u\}}H_i^l}\right) \wedge \left(\exists s \in S_{\underset{a}{\{u\}}H_i^l} : s = d_j\right), \tag{17}$$

where the first part of the statement ensures that the process is at the correct sub-holarchy; the second term checks whether the available capabilities can cover the requested testing parameters, and finally, the third statement assures that the dataset of the same family has been introduced to the holarchy in the training phase. If this statement yields false, then the holon informs its super-holon; otherwise, it sends the requests to its AlgH sub-holons and collects their answers to report to the super-holon. Algorithm 4 provides the details of the testing mechanism. It is worth mentioning that line 8 does not imply a blocking process in collecting the results. In practice, the requests are sent and are collected later based on the identity of the testing query.

ALGORITHM 4: The testing process.

Input: $q_j \left\langle id, (a_j, P_{a_j}), (d_j, P_{d_j}), O \right\rangle$

1 **Function** `Test`$(q_j \left\langle id, (a_j, P_{a_j}), (d_j, P_{d_j}), O \right\rangle)$

`// myself refers to the holon that calls this function`

2 results $\leftarrow \emptyset$;

3 **if** *AmIModel()* **then**

4 results $\leftarrow$ Perform("TEST", (d_j, P_{d_j}), O);

5 **else**

6 **if** $\left((name, a_j) \overset{\star}{\leq} (name, name_{\underset{a}{\{u\}}H_i^l})\right) \wedge \left(P_{a_j} \overset{\star}{\leq} C_{\underset{a}{\{u\}}H_i^l}\right) \wedge \left(\exists s \in S_{\underset{a}{\{u\}}H_i^l} : s = d_j\right) \vee (myself.LEVEL = 1)$ **then**

7 **foreach** $\underset{\{a,m\}}{\{i\}}H_k^{l+1} \in \underset{a}{\{u\}}H_i^l$ **do**

8 results $\leftarrow$ results $\bigcup$ Ask $\left(\underset{a}{\{i\}}H_k^{l+1}, \text{"TEST"}, q_i \left\langle id, (a_j, P_{a_j}), (d_j, P_{d_j}), O_i \right\rangle\right)$;

9 **end**

10 **end**

11 **end**

12 **return** Inform(myself.SUPER, "RESULTS", results);

13 **End Function**

A parametric general symbol is not limited to be used only in the algorithm specification of the query. To support the symbol for data specification, a process very similar to Algorithm 4 should be utilized to collect all the data first and pass them to the testing procedure. In other words, the data holons will collect the information of the datasets that match the query, instead of the testing results, and send them back to their super-holons recursively.

4 THEORETICAL ANALYSIS

Previous sections presented the details of the proposed distributed platform without providing a further evaluation of its correctness and efficiency. This section delves into the theoretical analysis of platform in terms of computational complexities and the correctness of the presented algorithms

4.1 Correctness

To make sure that the proposed platform is working correctly, we need to prove that all of its algorithms, i.e., training and testing, produce correct answers. We use soundness and completeness as the measures of correctness and, assuming the use of correctly implemented machine learning

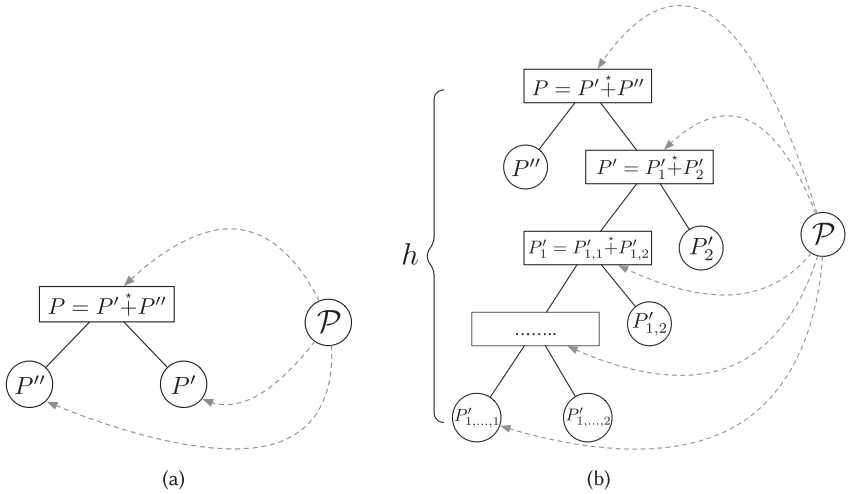


Fig. 5. The relationships between the parameter sets used in the lemmas and corollaries. The dashed lines represent the multiple ways that a query with configuration parameter set $\mathcal{P}$ is processed by each agent in the hierarchy.

algorithms and flawless datasets, we prove them for both of the proposed training and testing procedures. This is accomplished through the use of a series of lemmas and theorems presented in the rest of this section. Figure 5 depicts the relationship between the parameters assumed by the following lemmas and corollaries, and the detailed proofs are provided in Appendix A.

Lemma 4.1. *If $P, P', P'' \in \mathscr{P}$ such that $P' \neq P''$ and $P = P' \dot{+} P''$, where $\mathscr{P}$ is the set of all possible congruent sets of the same size, then for any non-general parametric set $\mathcal{P} \in \mathscr{P}$ (see Figure 5(a)), we have $\dot{\sim}(\mathcal{P}, P) < \dot{\sim}(\mathcal{P}, P')$ if and only if $|\mathcal{P} \cap P| < |\mathcal{P} \cap P'|$. In other words, this lemma ensures that the similarity ratio, $\dot{\sim}$, correctly assigns a greater value for the capability of an agent that is more specialized in handling an incoming query.*

Corollary 4.1.1. *If $P, P', P'' \in \mathscr{P}$ such that $P' \neq P''$ and $P = P' \dot{+} P''$, where $\mathscr{P}$ is the set of all possible congruent sets of the same size, then for any non-general parametric set $\mathcal{P} \in \mathscr{P}$ (see Figure 5(a)), only one of the statements $\dot{\sim}(\mathcal{P}, P) < \dot{\sim}(\mathcal{P}, P')$ or $\dot{\sim}(\mathcal{P}, P) < \dot{\sim}(\mathcal{P}, P'')$ will be true. To put it another way, according to this corollary, only one of the children of an agent node can have a similarity ratio larger than its parent. That is, a parent agent can safely stop processing the proposals from its subordinates as soon as it receives one that has a value, i.e., similarity ratio, greater than its own.*

Lemma 4.2. *If $P, P', P'' \in \mathscr{P}$ such that $P' \neq P''$ and $P = P' \dot{+} P''$, where $\mathscr{P}$ is the set of all possible congruent sets of the same size, then for any parametric set $\mathcal{P} \in \mathscr{P}$ (Figure 5(a)), if $\mathcal{P} \dot{\leq} P'$ and/or $\mathcal{P} \dot{\leq} P''$, then $\mathcal{P} \dot{\leq} P$.*

Corollary 4.2.1. *If in $P = P' \dot{+} P''$, any of the sets on right-hand side are the result of recursively applying operator $\dot{+}$ on two or more other parametric sets (Figure 5(b)), i.e.,*

$$P = \left(P'_1 \dot{+} P'_2\right) \dot{+} P'' = \left(\left(P'_{1,1} \dot{+} P'_{1,2}\right) \dot{+} P'_2\right) \dot{+} P'' = \ldots$$
$$= \left(\left(\ldots \left(P'_{\underbrace{1,\ldots,1}_{h}} \dot{+} P'_{\underbrace{1,\ldots,1,2}_{h}}\right) \dot{+} \ldots\right) \dot{+} P'_2\right) \dot{+} P'', \tag{18}$$

then

$$\mathcal{P} \overset{\star}{\leq} P' \underbrace{1, \ldots, 1}_{h} \Longrightarrow \mathcal{P} \overset{\star}{\leq} P. \tag{19}$$

That is, if there is at least one atomic/leaf agent at the bottom of the hierarchy that can fulfill a received query, then the request will be properly directed to that agent through the internal agent nodes.

Corollary 4.2.2. *If $P, P', P'' \in \mathscr{P}$ such that $P' \neq P''$ and $P = P' \overset{\star}{+} P''$, where $\mathscr{P}$ is the set of all possible congruent sets of the same size, then for any parametric set $\mathcal{P} \in \mathscr{P}$ (Figure 5(a)), if $\mathcal{P} \overset{\star}{\nleq} P'$ and $\mathcal{P} \overset{\star}{\nleq} P''$ then $\mathcal{P} \overset{\star}{\nleq} P$. In other words, if there is no chance that a received query be fulfilled by an agent at the bottom of the hierarchy, then none of the parent agents will be able to fulfill it either. Hence, the query will be blocked as early as possible by the parents.*

Theorem 4.3 (Soundness and Completeness of the Training Algorithm). *Giving the training algorithm all the information it needs to operate, it will provide a correct result whenever there exists one and a proper warning otherwise.*

Proof. Taking another deep look at the training algorithm, we notice that it always ends with adding new holons (data/algorithm/model) when it is needed, and returning the training results to the SYS holon. Consequently, to prove the soundness and completeness of the proposed method, we just need to prove that, first, the first pass of the method will only add the components if they are not already in the holarchy (no duplication); and, second, the final training command is correctly directed to the model and data holons so they start the fitting procedure.

To prove the first claim, let us assume the first pass will result in duplicate holons of the same settings at different parts of the holarchy. Based on the fact that the capability of a holon at any node is a parametric sum of all of its non-model sub-holons, duplication might occur if at any holon above the current holon in the holarchy, the training algorithm, starting at line 18, makes a wrong choice and directs the query to a wrong sub-holon. Since the proposals are made based on calculating the parametric similarity ratio, and also according to Lemma 4.1 and Corollary 4.1.1, it is guaranteed that this will not happen. That is, the holon's choice to forward the training query will always be correct, leading to inserting any new holon at its best place and preventing duplication during the first pass of the training procedure.

The second claim of this proof is guaranteed by code. As it was discussed in the details of the second pass of the training algorithm, any new structural changes in the holarchy, as the result of new training queries, is followed by updating all the references in the memory of the holons at the vicinity of the change (see Appendix C.2). □

Theorem 4.4 (Soundness and Completeness of the Testing Algorithm). *Giving the testing algorithm all the information it needs to operate, it will provide a correct result whenever there exists one and a proper warning otherwise.*

Proof. As it can be found out in the testing algorithm, the key decision at any holon of the holarchy is made based on Equation (17). Therefore, to show that the testing algorithm is sound and complete, we must prove that Equation (17) properly determines whether the holarchy is capable of answering the query. On account of the training steps, all of the holons with level ≥ 2 in the holarchy share the same name. As a result, at any node, if the name of the query algorithm is not the same as the name of the holon (the first part of Equation (17), then it means there will be no answer through that holon, thus the holon blocks the flow of the query to its sub-holons and returns an empty set as the response. Likewise, according to Equations (8), (9), and (10), the skills(the trained data) of any AlgH holon is the union of all of its sub-holons' skills. Consequently,

if the skills of holon do not satisfy the requirements of the query, then the third part of the equation makes the holon properly respond to the super-holon.

In case that both name and skill checks pass, the final decision is made by the second part of Equation (17). According to the training algorithm, the capabilities of any holon at level ≥2 of the holarchy is the parametric sum of the capabilities of its sub-holon. By Lemma 4.2 and Corollary 4.2.1, however, the holon will forward the test query to its sub-holons if any of its accessible subordinate atomic holons is capable of performing the test. Hence, the algorithm will find the right destination to execute the testing operation and return the appropriate result. Furthermore, pursuant to Corollary 4.2.2, the testing process will be stopped and returned, suitably utilizing the aforementioned equation. Hence, it would be impossible for the testing method to return the wrong result. □

4.2 Complexity

This section discusses the computational complexity of the proposed platform based on space and computational time criteria. This is done for both training and testing algorithms separately.

4.2.1 Training Algorithm. The training algorithm has two vertical passes in the holarchy. Since the passes in each of the data and algorithm sub-holarchies are carried out in parallel, we first take one of them into consideration and model it as a tree, in which the sub-holons of a holon are forming the children of its corresponding node. Figure 6 depicts two trees that represent the example holarchies that we delve into in our analysis. Let us assume that the maximum number of children (sub-holons) that an algorithm/data node in such a tree has is denoted by b_a and b_d, respectively. Similarly, assume that the current number of the leaf nodes (atomic holons) in each of the algorithm/data sub-trees is, respectively, shown by n_a and n_d. In the worst-case scenario, during the first pass, the holons need to wait for all of their sub-holons' proposals before they choose the best one. This scenario is similar to the one in which we check all of the children of a tree node before expanding the last one, as depicted by arrows in each picture of Figure 6. For the sake of brevity, let us focus on the algorithm sub-tree first. In a complete tree (Figure 6(a)), the height will be:

$$h_a = \log_{b_a} n_a. \tag{20}$$

Since, at each level, we check b_a nodes, the time complexity of the first pass of the training algorithm will be:

$$O(b_a \cdot h_a) = O(b_a \cdot \log_{b_a} n_a). \tag{21}$$

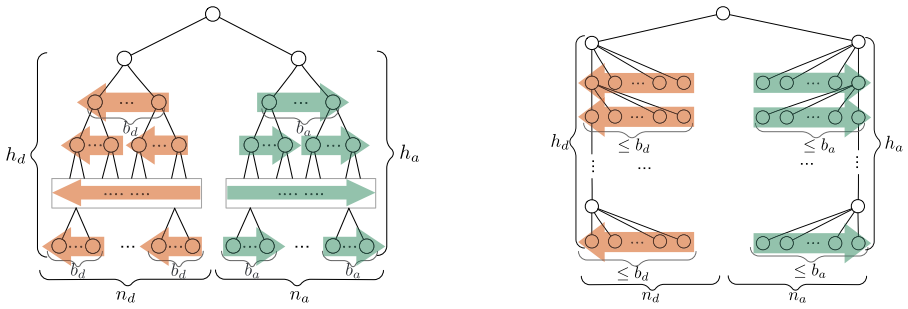
In an extreme case where $b_a - 1$ nodes of each level are terminal, the height would be:

$$h_a \frac{n_a - b_a}{b_a - 1} + 1 = \frac{n_a - 1}{b_a - 1}, \tag{22}$$

and therefore, the complexity of the first pass becomes:

$$O\left(b_a \cdot \frac{n_a - 1}{b_a - 1}\right) = O(n_a). \tag{23}$$

Similarly, both of above-mentioned tree layout on the data sub-tree will yield $O(b_d \cdot \log_{b_d} n_d)$ and $O(n_d)$, respectively. Regarding the second pass of the algorithm, since it only needs to follow the addresses without further checking the children, the time complexity for the same tree layouts would be $O(\log_{b_a} n_a)$ and $O(n_a)$ for the algorithm sub-tree and $O(\log_{b_d} n_d)$ and $O(n_d)$ for the data sub-tree, respectively. Considering the deepest tree layout (Figure 6(b)) and the fact that the second pass starts after the first pass finishes, the worst-case time complexity of the training algorithm



(a) a complete tree (b) a deep tree

Fig. 6. Tree representations of two extreme hierarchical structures. (a) depicts the case in which each parent has the same number of subordinates, and (b) represents the scenario in which only of the nodes at each level has a subordinate. In both of the figures, the arrows are assumed to be the order that the nodes are processed in each level.

for the algorithm sub-holarchy, would be:

$$O(n_a + n_a) = O(n_a). \tag{24}$$

Finally, taking both of DATA and ALG sub-holarchies into consideration and recalling the fact that training at each sub-holarchy is run in parallel, the worst-case time complexity of the training algorithm would be:

$$O(\max(n_a, n_d)). \tag{25}$$

Assuming a holarchy with n_a and n_d atomic algorithm and data holons, respectively, at any time that we run the training query, the worst-case scenario for the space complexity occurs when there is no holon representing neither the queried algorithm nor dataset. In this case, in addition to using a fixed amount of memory in each holon, five new holons are created: two super-holons, two new holons for the new algorithm and dataset, and one model holon. Therefore, the space complexity in terms of the number of new entities would be $O(1)$. However, the amount of the memory that is used by the holons during the passes of the training process is $O(\max(n_a, n_d))$, as it uses a fixed amount of memory for each step of the procedure until it creates the holon and begins the procedure. It is important to note that we have ignored the complexities of the data-mining algorithm and used datasets in our calculations.

4.2.2 Testing Algorithm. The computational complexity analysis of the testing algorithm is very similar to that of the training process. Again, let us assume that b_a, b_d, n_a, and n_d, respectively, denote the maximum branching factor of a node in algorithm and data sections, and the total number of leaf nodes representing the trained algorithms and the stored datasets in the corresponding tree model of the holarchy. Regarding the structure, we make the same hypothesis that we made before and use the layout depicted in Figure 6. For a query with non-general parameters, the first step of the testing algorithm is to locate the testing dataset and get its address. This needs two travels in the depth of the DATA sub-holarchy to find the dataset and return its address, and two travels in the ALG portion and return the results. In case of a complete tree in each of DATA and ALG sections, these can be done in $O(b_d \cdot \log_{b_n} n_d)$ and $O(b_a \cdot \log_{b_a} n_a)$ time, respectively. As we need to finish the data search process before we start testing and traveling in the ALG tree, the total complexity for such a tree would be

$$O(b_d \cdot \log_{b_d} n_d + b_a \cdot \log_{b_a} n_a). \tag{26}$$

However, if only one of the nodes in each level is expanded and holds children, then the height of the each DATA and ALG sub-holarchies will be $\frac{n_d-1}{b_d-1}$ and $\frac{n_a-1}{b_a-1}$, respectively, and the worst-case required time for the passes in each will be $O(\frac{n_d-1}{b_d-1})$ and $O(\frac{n_a-1}{b_a-1})$, accordingly. As a result, the overall testing complexity becomes

$$O\left(b_d \cdot \frac{n_d-1}{b_d-1} + b_a \cdot \frac{n_a-1}{b_a-1}\right) = O(n_d + n_a). \tag{27}$$

During the testing procedure, no new holon is created, but the existing ones are used. As a result, the space complexity of the testing operation will be the amount of the memory that is used by each holon. This amount is fixed, i.e., $O(1)$, and for each data or algorithm component. Consequently, the space for the entire testing process will be solely a function of the number of steps in the process. In other words, for the same structural settings that we discussed above, the worst-case space complexity of the entire procedure will be $O(b_d \cdot \log_{b_d} n_d + b_a \cdot \log_{b_a} n_a)$ and $O(b_d \cdot \frac{n_d-1}{b_d-1} + b_a \cdot \frac{n_a-1}{b_a-1}) = O(n_d + n_a)$.

As it was stated in the beginning, the above-mentioned analysis of the testing method is for the case that a non-general query is sent to the system. As the proposed testing operation is capable of carrying out multiple sub-tasks at once, thanks to the intrinsic distributed property of multi-agent systems, we expect to save time conducting general tests. For instance, assume that in the worst case the query is intended to test all the algorithms, i.e., $\Lambda = \{(\star, \{\star\})\}$, on all the available datasets, i.e., $\Delta = \{(\star, \{\star\})\}$. Furthermore, let us hypothesize that, being of the same type each, all the existing algorithms have been previously trained on all the available datasets. With the same notations as before, such a holarchy will have $n_d \cdot n_a$ model holons in addition to n_d atomic data and n_a atomic algorithm holons. The above-mentioned general test query on such a holarchy can be decomposed to $n_d \cdot n_a$ non-general test queries, and if we feed them to the system separately, then the time complexity for each of the mentioned complete and non-complete holarchical structures will be

$$O(n_d \cdot n_a(b_d \cdot \log_{b_d} n_d + b_a \cdot \log_{b_a} n_a)) \tag{28}$$

and

$$O\left(n_d \cdot n_a \left(b_d \cdot \frac{n_d-1}{b_d-1} + b_a \frac{n_a-1}{b_a-1}\right)\right) = O({n_d}^2 \cdot n + n_d \cdot {n_a}^2) \tag{29}$$

in the given order. However, having a holarchy with all its algorithms trained on all its data, implies that $n_d \leq n_a$. Therefore, the worst-case complexities will become $O({n_a}^2(b_a \cdot \log_{b_a} n_a))$ and $O({n_a}^3)$, respectively. In contrast, being able to process all the queries at once, the proposed testing method will employ all of the holons in its structure to process such a general query in parallel. Let us consider the complete holarchy structure. The total number of holons in each algorithm and data sections will be the sum of the number of atomic and composite holons in each part. Taking the number of trained models into the account, the total number of holons in the holarchy, except the SYS, DATA, and ALG will be:

$$n_d \cdot n_a + \sum_{i=0}^{\log_{b_d} n_d} \frac{n_d}{(b_d)^i} + \sum_{i=0}^{\log_{b_a} n_a} \frac{n_a}{(b_a)^i} = n_d \cdot n_a + \frac{b_d \cdot n_d - 1}{b_d - 1} + \frac{b_a \cdot n_a - 1}{b_a - 1}. \tag{30}$$

Consequently, based on the fact that in such a dense holarchy $n_d \leq n_a$, the worst-case time complexity to process the given general test query will be $O({n_a}^2)$, which is less than the aforementioned $O({n_a}^2(b_a \cdot \log_{b_a} n_a))$. Likewise, for the non-complete holarchical tree that we used in previous analyses, on account of the fact that the total number of holons is

$$n_d \cdot n_a + b_d \cdot \frac{n_d-1}{b_d-1} + b_a \cdot \frac{n_a-1}{b_a-1}, \tag{31}$$

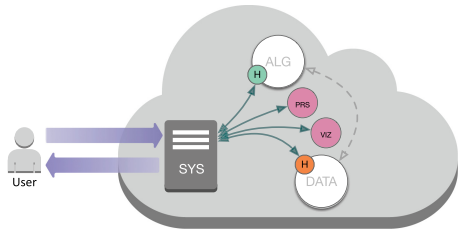


Fig. 7. The initial components of the experiment.

the worst-case time complexity remains the same, that is, $O({n_a}^2) < O({n_a}^3)$.

5 EXPERIMENTAL EVALUATION

Our presented distributed machine learning model is not bounded by a specific multi-agent development framework. In fact, each module and section of the entire system can be developed using different tools; and as long as they keep following the suggested protocols and training/testing procedures, the system would demonstrate the expected behavior. For the sake of experimentation, we used **Smart Python Agent Development Environment (SPADE)**[1] [33] to implement our platform.[2] This is mainly due to its simple yet flexible model and the abundance of machine learning and data-mining libraries available in Python programming language. Appendix D.1.1 provides a brief overview of the SPADE architecture.

The initial high-level outline of the the test environment, as depicted in Figure 7, is composed of the SYS, DATA, and ALG holons together with two additional PRS and VIZ agents that have the responsibilities of processing the received queries and generating plots for the results, respectively. Additionally, the User agent depicted in this schema simulates the role of an external human agent and basically acts as an automatic query generator. The queries sent to the system are handed over the PRS agent to validate the basic query structures and inform about any potential errors before beginning the task. As soon as the result tensors are available after conducting a machine learning task, they are sent to the VIZ agent to plot and deliver them according to a pre-specified format. The details of using SPADE to construct the HAMLET-based experimental environment are provided in Appendix D.1.2.

The experiment makes use of eight classification, eight regression, and eight clustering algorithms being trained and tested on nine standard datasets. All of the used algorithms are from the scikit-learn [52] library, and their details are listed in Appendix D, Tables 8 and 9, respectively. Last but not least, all the experiments have been carried out in a PC with Intel Core-i5 @1.6 GHz CPU and 16 GB RAM running Ubuntu OS and Python 3.7. Please note that for the sake of saving the space of the main body of the article, we present the empirical results in form of tables. The readers interested in the visual outcomes generated by the VIZ agent are encouraged to refer to Appendix D.3.2.

5.1 Training

The first set of queries pertains to training the aforementioned classification, regression, and clustering algorithms on the available datasets, and finally adding test datasets to the system. For the sake of creating a high-loaded experiment, we trained all the algorithms on all of their

[1]SPADE's source code is available at https://github.com/javipalanca/spade.
[2]The source code of our implementation of HAMLET and the experiments are available at https://github.com/aesmaeili/HAMLET.

Table 1. Accuracy and Time of Training Each Classification Algorithm on a Specific Dataset

Measure	Alg.	Dataset					
		breast cancer	digits	iris	art. class.	art. moon	wine
classification accuracy | training time(sec.)	A01	**0.968**|1.373	**1.000**|0.028	**0.989**|0.001	**0.611**|0.016	**0.887**|0.002	**0.991**|0.122
	A02	**0.440**|0.010	**0.918**|0.106	**0.378**|0.001	**0.344**|0.032	**0.590**|0.006	**0.170**|0.002
	A03	**0.968**|0.010	**0.999**|0.073	**0.700**|0.001	**0.344**|0.019	**0.517**|0.003	**0.849**|0.002
	A04	**1.000**|0.010	**1.000**|0.072	**0.978**|0.001	**0.561**|0.018	**0.870**|0.003	**1.000**|0.002
	A05	**0.868**|0.007	**0.972**|0.169	**0.978**|0.001	**0.944**|0.024	**0.920**|0.004	**0.925**|0.002
	A06	**0.886**|0.001	**0.823**|0.003	**0.700**|0.001	**0.513**|0.002	**0.840**|0.001	**0.660**|0.001
	A07	**1.000**|0.006	**1.000**|0.014	**1.000**|0.001	**1.000**|0.009	**1.000**|0.001	**1.000**|0.001
	A08	**0.880**|0.001	**0.913**|0.002	**0.922**|0.001	**0.504**|0.001	**0.850**|0.001	**0.717**|0.001

corresponding datasets. The holarchical structures resulted from the resource addition and algorithm training tasks are discussed in Appendix D.3.1.

The training results for the classification, regression, and clustering are presented, respectively, in Tables 1, 2, and 3 (and visually in Figures 13, 14, and 15). Each row in each table provides the performance of training an ML algorithm on the available training datasets. For consistency reasons, we have used the same naming convention as the ones in Appendix D.2. In the training queries, we have used *accuracy*, *mean squared error*, and intrinsic *Fowlkes-Mallows score* [29] as the measures of performance for classification, regression, and clustering tasks, respectively. Each of these measures alongside the amount of time that the training procedure for each algorithm has taken is reported in a separate table. The reported results provide various analytical and comparative insights about the training procedure. For instance, in Table 1, the classification algorithms *A04* and *A07* are among the most efficient and effective algorithms trained on the *iris* dataset. This is because of achieving the highest accuracy in the lowest amount of time. Likewise, algorithm *A03* is the least efficient and effective algorithm trained on the *artificial moon* dataset due to its relatively high training time and lowest accuracy score. Similar information can also be obtained from studying the reported results for the other types of conducted machine learning tasks. For example, algorithms *A11* and *A13* are among the best regression algorithms that are included in the platform because of their relatively lower error and training time on all of the available datasets (see Table 2). However, the clustering algorithm *A24* is one of the fast algorithms that yields a relatively higher performance measure on most datasets. Please, note that all the parameters during training, evaluating their results, and so on, are chosen based on no specific reason but as an example to demonstrate the capability of the proposed platform.

5.2 Testing

The algorithms and datasets have been selected in such a way that we could carry out complex testing queries on the platform. This section demonstrates some of the testing capabilities that the proposed platform provides, though its flexibility is not limited to only those listed here. The demonstrations are on classification, regression, and clustering tasks, and we still use tables to present the results, with supplementary visual representation in Appendix D.3.2. It is worth noting that the difference between testing and training a clustering algorithm in this article is that training allows configuring the algorithm through its parameters, whereas testing merely looks for the algorithm based on the provided criteria and runs it on the specified data.

In the first set of testing queries, the user agent requests the results of testing all the algorithms that have kernel parameter with value "rbf" on all the inserted test datasets. Please note that based on the assumption that we have made earlier, the algorithms will be tested on the datasets

Table 2. Mean Squared Error and Time of Training Each Regression Algorithm on a Specific Dataset

Measure	Alg.	Dataset: boston	Dataset: diabetes	Dataset: art.regr.
regression error(mse.) | training time(sec.)	A09	**19.923**|0.001	**2,681.173**|0.001	**0.008**|0.001
	A10	**23.572**|0.001	**26,884.843**|0.001	**3.384**|0.001
	A11	**20.058**|0.001	**3,084.411**|0.001	**0.874**|0.001
	A12	**23.572**|0.009	**26,884.843**|0.003	**3.384**|0.002
	A13	**21.237**|0.001	**2,731.674**|0.001	**0.118**|0.001
	A14	**64.885**|0.008	**5,358.552**|0.005	**37,387.087**|0.002
	A15	**89.873**|0.003	**6,294.532**|0.002	**37,915.425**|0.001
	A16	**24.215**|0.002	**6,299.868**|0.001	**4,356.211**|0.001

Table 3. Fowlkes-Mallows Score and Time of Running Each Clustering Algorithm on a Specific Dataset

Measure	Alg.	Dataset: breast cancer	Dataset: digits	Dataset: iris	Dataset: art. class.	Dataset: art. moon	Dataset: wine
clustering score (FM) | training time(sec.)	A17	**0.792**|0.061	**0.700**|0.423	**0.821**|0.036	**0.372**|0.203	**0.696**|0.039	**0.584**|0.034
	A18	**0.792**|0.099	**0.696**|0.590	**0.821**|0.049	**0.374**|0.381	**0.696**|0.058	**0.584**|0.056
	A19	**0.469**|0.039	**0.579**|0.086	**0.636**|0.034	**0.238**|0.056	**0.468**|0.029	**0.391**|0.031
	A20	**0.729**|0.010	**0.315**|0.084	**0.705**|0.002	**0.443**|0.048	**0.706**|0.008	**0.581**|0.002
	A21	**0.729**|0.011	**0.315**|0.083	**0.582**|0.002	**0.577**|0.050	**0.706**|0.009	**0.581**|0.003
	A22	**0.729**|0.020	**0.315**|0.142	**0.573**|0.002	**0.577**|0.043	**0.706**|0.018	**0.581**|0.004
	A23	**0.671**|0.216	**0.493**|0.330	**0.751**|0.008	**0.414**|0.049	**0.706**|0.011	**0.582**|0.014
	A24	**0.739**|0.013	**0.817**|0.164	**0.822**|0.002	**0.367**|0.027	**0.733**|0.009	**0.582**|0.002

on which they have been trained before. Hence, we expect to see the results accordingly. This query is translated to $\Lambda = \{(\star, \{(\text{kernel, rbf})\}\}, \Delta = \{(\star, \{(\text{type, test})\})\}$, and O = {format=plot, measures=[accuracy, mean square error]}, and the results are presented in Table 4 (also Figure 16). As it can be seen, platform has correctly determined the proper algorithms with their corresponding measures for each dataset. For instance, among eight regression algorithms that are defined in the system, HAMLET has correctly identified *A14* and *A15* as the ones that have "rbf" kernel. Similarly, algorithms *A03*, *A04*, and *A05* are the only classification algorithms that match the criteria of the query. Additionally, the platform has successfully tested the algorithms on the datasets that they had been trained on despite the fact that we have not explicitly specified which datasets should be used for the tasks. This capability of the platform in automatically determining how algorithms, datasets, and measures should be used side-by-side, together with the analytical and comparative insights that it provides, is particularly noteworthy for the cases in which users are unaware of the available resources and try to let the system make proper choices based on the most recent resources.

The second experimented query narrows the testing procedure and determines the results of applying all the SVC algorithms on the breast cancer dataset, based on accuracy and area under the ROC curve [23] scores. The query configuration for this test is as follows: $\Lambda = \{(\text{SVC}, \{\star\})\}, \Delta = \{(\text{breast cancer}, \{(\text{type, test})\})\}$, and O = {format=plot, measures=[accuracy, roc-auc]}, and the generated results are given in Table 5. As it can be seen, the subordinate agents of HAMLET have determined algorithms *A01*, *A02*, *A03*, and *A04* as the ones that implement SVC, and reported their

Table 4. The Results of Testing All Classification and Regression Algorithms that Have Parameter *kernel* Equal to *rbf* on All Proper Datasets, in Terms of Accuracy and Mean Squared Error (Mse.)

		Dataset								
Measure	Alg.	boston	breast cancer	diabetes	digits	iris	art. class.	art. moon	art. regr.	wine
accuracy	A03	–	0.943	–	0.993	0.600	0.314	0.475	–	0.708
	A04	–	0.921	–	0.987	0.933	0.517	0.825	–	0.750
	A05	–	0.886	–	0.961	0.917	0.589	0.870	–	0.833
mse.	A14	63.654	–	4,529.860	–	–	–	–	35,326.226	–
	A15	91.531	–	5,289.042	–	–	–	–	36,858.398	–

Table 5. The Results of Testing All SVC Algorithms with Any Parameter on the Breast Cancer Dataset, in Terms of Accuracy and Area Under the ROC Curve Score

		Algorithm			
Measure	Dataset	A01	A02	A03	A04
accuracy score	breast cancer	0.974	0.469	0.943	0.921
roc auc score	breast cancer	0.972	0.408	0.945	0.922

Table 6. The Results of Testing All SVC Algorithms with Any Parameter on the Breast Cancer Dataset, in Terms of Accuracy and Area Under the ROC Curve Score

		Algorithm							
Measure	Dataset	A01	A02	A03	A04	A05	A06	A07	A08
accuracy score	breast cancer	0.835	0.665	0.475	0.825	0.870	820	0.955	0.825
		A17	A18	A19	A20	A21	A22	A23	A24
roc auc score	breast cancer	0.310	0.310	0.724	0.000	0.000	0.000	0.000	0.420

test performances using two measures. Please note that the slightly different representation of the tables are for the sake of saving space, and the corresponding plot generated by the VIZ agent is presented in Appendix D.3.2, Figure 17.

The third experimental testing query evaluates all the trained algorithms on the artificial moon dataset based on three measures of accuracy, mean squared error, and homogeneity score [59]. This test is conducted based on the following settings: $\Lambda = \{(\star, \{\star\})\}$, $\Delta = \{(\text{moon}, \{(\text{type}, \text{test})\})\}$, and $O = \{\text{format=plot, measures=[accuracy, mean squared error, homogeneity]}\}$. Please note that we have hypothetically considered that we are not sure whether the moon datasets is of classification or regression type, so we have specified a measure from each machine learning task and let the platform choose the proper one itself. The obtained results are presented in Table 6 (also Figure 18). As it can be seen, only accuracy and homogeneity scores are reported due to the fact that artificial moon is a nominal dataset suitable for classification/clustering tasks.

Finally, the fourth experiment pertains to testing all of the available algorithms on all of the available test datasets, in terms of the same three measures: accuracy, mean squared error, and homogeneity. This test employs all of the resources in the holarchy to process the query concurrently. The query configuration is: $\Lambda = \{(\star, \{\star\})\}$, $\Delta = \{(\star, \{(\text{type}, \text{test})\})\}$, and $O = \{\text{format=plot, measures=[accuracy, mean squared error, homogeneity]}\}$, and the results are presented in Table 7

Table 7. The Results of Testing All Algorithms with Any Parameter on All the Test Datasets, in Terms of Accuracy, Clustering Homogeneity Score, and/or Mean Square Error

		Dataset								
Measure	Alg.	boston	breast cancer	diabetes	digits	iris	art. class.	art. moon	art. regr.	wine
classification accuracy	A01	–	0.974	–	**0.975**	0.967	0.556	0.835	–	0.958
	A02	–	0.469	–	**0.900**	0.267	0.314	0.665	–	0.181
	A03	–	0.943	–	**0.993**	0.600	0.314	0.475	–	0.708
	A04	–	0.921	–	**0.987**	0.933	0.517	0.825	–	0.750
	A05	–	0.886	–	**0.961**	0.917	0.589	0.870	–	0.833
	A06	–	**0.912**	–	0.764	0.617	0.481	0.820	–	0.708
	A07	–	0.908	–	0.811	0.950	0.556	**0.955**	–	0.931
	A08	–	**0.904**	–	0.897	0.867	0.478	0.825	–	0.736
clustering homogeneity	A17	–	0.476	–	**0.695**	0.686	0.066	0.310	–	0.435
	A18	–	0.476	–	**0.721**	0.686	0.071	0.310	–	0.435
	A19	–	0.727	–	0.573	**0.902**	0.071	0.823	–	0.536
	A20	–	0.000	–	0.000	**0.536**	0.009	0.000	–	0.000
	A21	–	0.000	–	0.000	**0.111**	0.000	0.000	–	0.000
	A22	–	**0.000**	–	0.000	-0.000	0.000	0.000	–	0.000
	A23	–	**0.696**	–	0.323	0.652	0.043	0.000	–	0.432
	A24	–	0.464	–	**0.763**	0.610	0.013	0.420	–	0.432
regression mean squared error	A09	25.790	–	3,233.154	–	–	–	–	0.013	–
	A10	25.833	–	26,336.766	–	–	–	–	5.026	–
	A11	25.830	–	3,128.198	–	–	–	–	1.390	–
	A12	25.833	–	26,336.766	–	–	–	–	5.026	–
	A13	26.812	–	3,150.794	–	–	–	–	0.167	–
	A14	63.654	–	4,529.860	–	–	–	–	35,326.226	–
	A15	91.531	–	5,289.042	–	–	–	–	36,858.398	–
	A16	30.584	–	5,220.941	–	–	–	–	5,788.403	–

(also Figure 19). This test shows how HAMLET can be used to comprehensively explore all the available models using a single query. Recalling the number of resources we have inserted and trained, one can easily validate that the results demonstrated in this figure are complete. That is, there is no algorithm/dataset that had been trained before but missed here. This is worth noting that this query deliberately uses a clustering measure, i.e., homogeneity score, from the one we have used before, i.e., Fowlkes-Mallows (see Table 3), to demonstrate how the agents representing the models appropriately respond to the query based on their capabilities.

The presented experiments demonstrate the flexibility and capabilities of the proposed platform on performing machine learning tasks. Please note that the platform does not make any optimization on the parameters of the results of the tasks, except the ones that are already employed by each algorithm. Consequently, any poor performance is solely the result of poor underlying utilized algorithms and not because of the platform. The highlights and font decorations in Table 7 endeavor to exhibit the analytical possibilities that our platform provides for the classification and clustering tasks based on a selected set of performance measures. The underlined boldface numbers specify the maximum performance that corresponding algorithms have achieved on all datasets. The color highlights, however, accentuate the best algorithms, in terms of the specified performance measures, for conducting a classification or clustering task on a specific dataset. We have used different colors for classification and clustering tasks to help distinguish them from each other. The following are some of the example intuitions that can be made based on the results:

- Algorithm *A01* is the best available classification algorithm for the *breast cancer* dataset.
- The majority of the classification algorithms, i.e., A01–A05, have achieved their highest performances on the *digit* dataset.
- The *A19* clustering algorithm has performed better than the other algorithms in terms of the number of the datasets that it resulted the highest homogeneity score.
- The *make classification* dataset has been the most challenging dataset for both available classification and clustering algorithms. This is because of the relatively low scores reported.

6 CONCLUSION

In this article, we presented a hierarchical multi-agent platform for the management and execution of data-mining tasks. The proposed solution models a machine learning problem as a hypergraph and employs autonomous agents to cooperatively process and answer training and testing queries based on their innate and learned capabilities. Using an agent-based approach for the problem, on one hand, facilitates the deployment of the system on distributed infrastructures and computer networks, and on the other hand, provides the researchers with the flexibility and freedom of adding their own machine learning algorithms or datasets with customized behaviors. The platform provides numerous potential benefits for both research and deployment purposes. It can be used by research communities to share, maintain, and have access to the most recent machine learning problems and solutions, such that they are able to analyze new data, compare the performance of new solutions with the state-of-the-art. It can also be utilized in devising new solutions by letting the designers experiment with different versions of their methods distributedly and in parallel to understand the behavior of their algorithms under different configurations and select the most appropriate one accordingly.

We have assessed the proposed platform both theoretically and empirically. By means of a set of theorems and lemmas, we proved that the agent-based solution is sound and complete. That is, given a machine learning query, it always returns the correct answer whenever one exists and warns appropriately otherwise. We have also analyzed its performance in terms of time complexity and space requirements. According to the discussions, our proposed method requires polynomial time and memory to respond to training and testing queries in the worst case. Furthermore, we designed and carried out a set of experiments to show the flexibility and capabilities of the proposed agent-based machine learning solution. We used 24 classification, regression, and clustering algorithms and applied them to 9 real and artificial datasets. The results of both training and testing queries, plotted by the system, demonstrated its correctness together with how a user can perform single and batch queries to extract the existing knowledge.

This article proposed the foundations of the suggested agent-based machine learning platform and can be extended in various ways to support new applications and scenarios. At its presented state, tasks such as visualization and pre-processing are managed and conducted by a separate unit and the data agents, respectively, whereas they can be performed by separate hierarchies to make sophisticated results and data preparation services available to the system. To support sophisticated algorithms and analyses, the platform can also be expanded to support machine learning pipe-lined tasks through more horizontal cooperative interactions between the agents at different levels. In the presented version, the structure grows because of the training process and adding new algorithms/datasets. This dynamic growth property can be enhanced even more by letting already-built substructures merge together. Last but not least, the dynamic behavior of the platform can be improved by exclusively handling abnormal events such as changes to the structure because of agent permanent failures. We are currently working on these extensions and suggest them as future work.

APPENDIX

A PROOFS

A.1 Lemma 4.1

Proof. Assuming $\mathcal{P} = \{(\rho_i, v_i\}$, let's define $L = \{(\rho_i, v_i) \in P : v_i \neq v_i \wedge (v_i = \star \veebar v_i = \star)\}$, $M = \{(\rho, v_i) : v_i \neq v_i \wedge (v_i \neq \star \wedge v_i \neq \star)\}$, and L' and M' sets similarly. Since $\forall P, P' \in \mathscr{P}$, $P \cap P' = \{(\rho, \nu) : (\rho, \nu) \in P, P'\}$ and as all the sets are congruent, $\overset{\star}{\sim}(\mathcal{P}, P) < \overset{\star}{\sim}(\mathcal{P}, P')$ means:

$$\begin{aligned}\sum_{\substack{v_i = v_i' \\ i}} \overset{\star}{\sim}((\rho_i, v_i),(\rho_i, v_i')) + \prod_{\substack{v_i \neq v_i' \\ i}} \overset{\star}{\sim}((\rho_i, v_i), (\rho_i, v_i')) \\ < \sum_{\substack{v_i = v_i'' \\ i}} \overset{\star}{\sim}((\rho_i, v_i), (\rho_i, v_i'')) + \prod_{\substack{v_i \neq v_i'' \\ i}} \overset{\star}{\sim}((\rho_i, v_i), (\rho_i, v_i''))\end{aligned} \tag{32}$$

$$\implies |\mathcal{P} \cap P| + \alpha^{|L|}\beta^{|M|} < |\mathcal{P} \cap P'| + \alpha^{|L'|}\beta^{|M|'}. \tag{33}$$

This needs that either $|\mathcal{P} \cap P| < |\mathcal{P} \cap P'|$, which proves the claim, or:

$$\begin{aligned}|\mathcal{P} \cap P| = |\mathcal{P} \cap P'| \quad &\implies \quad \alpha^{|L|}\beta^{|M|} < \alpha^{|L'|}\beta^{|M'|} \\ &\implies \alpha^{|L|-|L'|} < \beta^{|M'|-|M|}.\end{aligned} \tag{34}$$

However, due to Definitions 3.3 and 3.6, we must have $|L| + |M| = |L'| + |M'|$, $|L| < |L'|$, and $|M| > |M'|$. This requires that there are more general pairs in P than in P', which is not possible according to the Definition 3.5.

The second part of the proof is trivial, because of the fact that $|\mathcal{P} \cap P| > 1$ and $|\mathcal{P} \cap P'| > 1$ and as they are the dominant factors in the calculation of the parametric similarity ratio, we can write:

$$\begin{aligned}|\mathcal{P} \cap P| < |\mathcal{P} \cap P'| &\implies |\mathcal{P} \cap P| + \alpha^{|L|}\beta^{|M|} < |\mathcal{P} \cap P'| + \alpha^{|L'|}\beta^{|M|'} \\ &\implies \overset{\star}{\sim}(\mathcal{P}, P) < \overset{\star}{\sim}(\mathcal{P}, P').\end{aligned} \tag{35}$$

□

A.2 Corollary 4.1.1

Proof. Let us assume that both $\overset{\star}{\sim}(\mathcal{P}, P) < \overset{\star}{\sim}(\mathcal{P}, P')$ and $\overset{\star}{\sim}(\mathcal{P}, P) < \overset{\star}{\sim}(\mathcal{P}, P'')$ are true. According to Lemma 4.1, this means that $|\mathcal{P} \cap P| < |\mathcal{P} \cap P'|$ and $|\mathcal{P} \cap P| < |\mathcal{P} \cap P''|$. This means both of the following statements are true:

$$\exists(p_i, v_i' \neq \star) \in P' \quad : \quad (p_i, v_i') \in \mathcal{P} \wedge (p_i, v_i') \notin P \tag{36}$$

$$\exists(p_i, v_i'' \neq \star) \in P'' \quad : \quad (p_i, v_i'') \in \mathcal{P} \wedge (p_i, v_i'') \notin P. \tag{37}$$

There are two possible cases: (i) $v_i' = v_i''$ or (ii) $v_i' \neq v_i''$. According to Definition 3.5, the first case is not possible, because that will cause the corresponding pair to appear in set P as well. However, the second case means that we have two different values for the the same parameter in set $\mathcal{P}$, which contradicts the definition of parametric sets (Definition 3.1). Therefore, we conclude that the claim of the corollary is correct. □

A.3 Lemma 4.2

Proof. According to Definition 3.4, we have:

$$\mathcal{P} \overset{\star}{\leq} P' \implies \forall(\rho, \nu) \in \mathcal{P} : (\rho, \nu) \in P' or (\rho, \star) \in P'. \tag{38}$$

However, based on the membership of (ρ, v) in P'' and Definition 3.5, only one of the following cases will happen:

$$(\rho, v) \in P' \wedge (\rho, v) \in P'' \Longrightarrow (\rho, v) \in P \Longrightarrow \mathcal{P} \overset{\star}{\leq} P \tag{39}$$

$$(\rho, v) \in P' \wedge (\rho, v) \notin P'' \Longrightarrow (\rho, \star) \in P \Longrightarrow \mathcal{P} \overset{\star}{\leq} P \tag{40}$$

$$(\rho, \star) \in P' \wedge (\rho, v) \in P'' \Longrightarrow (\rho, \star) \in P \Longrightarrow \mathcal{P} \overset{\star}{\leq} P \tag{41}$$

$$(\rho, \star) \in P' \wedge (\rho, v) \notin P'' \Longrightarrow (\rho, \star) \in P \Longrightarrow \mathcal{P} \overset{\star}{\leq} P. \tag{42}$$

It can be seen that regardless of the membership of (ρ, v) in P'' the claim is proven. We can use a similar process for $\mathcal{P} \overset{\star}{\leq} P''$ and finally show the correctness of this lemma. □

A.4 Corollary 4.2.1

Proof. Based on the relationship between the sets, according to Equation (18), we can write:

$$\begin{aligned} \mathcal{P} \overset{\star}{\leq} P'_{\underbrace{1, \ldots, 1}_{h}} \Longrightarrow \mathcal{P} \overset{\star}{\leq} P'_{\underbrace{1, \ldots, 1}_{h-1}} \Longrightarrow \ldots \Longrightarrow \mathcal{P} \overset{\star}{\leq} P'_{1,1} \\ \Longrightarrow \mathcal{P} \overset{\star}{\leq} P'_1 \Longrightarrow \mathcal{P} \overset{\star}{\leq} P' \Longrightarrow \mathcal{P} \overset{\star}{\leq} P. \end{aligned} \tag{43}$$

□

A.5 Corollary 4.2.2

Proof. According to Definitions 3.4 and 3.5, we have:

$$\mathcal{P} \overset{\star}{\nleq} P', P'' \Longrightarrow \exists (\rho, v) \in \mathcal{P} : (\rho, v) \notin P', P'' \wedge (\rho, \star) \notin P', P'' \tag{44}$$

$$(\rho, v) \notin P', P'' \Longrightarrow (\rho, v) \notin P \textbf{ and } (\rho, \star) \notin P', P'' \Longrightarrow (\rho, \star) \notin P. \tag{45}$$

Consequently,

$$\exists (\rho, v) \in \mathcal{P} : (\rho, v) \notin P \wedge (\rho, \star) \notin P \Longrightarrow \mathcal{P} \overset{\star}{\nleq} P. \tag{46}$$

□

B AUXILIARY ALGORITHMS

Algorithms 5 and 6 are the helping algorithms that are extensively used by the primary holarchy construction algorithms.

ALGORITHM 5: The algorithm for joining a holon and updating the capability.

```
1  Function JoinHolon(new-super-holon)
       // myself refers to the holon that calls this function
2      Update("SUPER", new-super-holon);
3      Inform(new-super-holon, "CAPABILITY", myself.C);
4  End Function

5  Function UpdateCapability(new-capability)
       // myself refers to the holon that calls this function
6      if myself.LEVEL > 1 then // I am not ALG holon
7          myself.C ← myself.C [⋆ ∔] new-capability;
8          Inform(myself.SUPER, "CAPABILITY", myself.C);
9      end
10 End Function
```

ALGORITHM 6: The algorithm for spawning a model holon and propagating its address to the top of holarchy.

```
1  Function SpawnModel()
       // myself refers to the holon that calls this function
2      new_model ← CreateModel(myself.NAME, myself, myself.C,{});
3      InformAddress(qid_i,new_model.ADDRESS);
4  End Function

5  Function InformAddress(query-id, address)
       // myself refers to the holon that calls this function
6      if myself.LEVEL > 1 then // I am not ALG/DATA holon
7          path-list ← AddToFront(path-list, myself.ADDRESS);
8      end
9      StoreAddress(query-id, address);
10     Ask(myself.SUPER, "INFORM-ADDRESS",
11                                   query-id, myslef.ADDRESS);
12 End Function
```

C ADDITIONAL EXAMPLES

C.1 Algorithm Addition Example

Figure 8 demonstrates a step-by-step process of Algorithms 1 and 2 in a simple example case. For the sake of clarity, we have shown only the names and the values of the configuration parameters for each input algorithm (in red color). Furthermore, the identity and the name of the created holons are given in *id:name* format inside of the nodes to help readers understand the order of the created holons. In part 8(a) of this figure, the ALG holon is asked to add algorithm X with parameter values $\{a, b, c, d\}$ to its holarchy. Since ALG does not have any subs, a new holon is created as its sub-holon to represent algorithm X (part 8(b)). When the system is asked to add algorithm Y with parameter values $\{o, p, q\}$, ALG calls for proposal from its sub-holons, *1:X* in this example, and, since the proposal value is not 0 (due to the dissimilarity of its name), the new sub-holon *2:Y* is created. In part 8(c), the resulted holarchy is requested to add a new algorithm with name X and parameter values $\{a, e, c, d\}$. First, ALG locates the sub-holon that pertains to the algorithms of name X and then forwards the algorithm specifications to that holon. Upon receiving the request, holon *1:X* calculates the parametric similarity ($\overset{*}{\sim}$ between its capabilities and the parameter set of the algorithm). The value $\boxed{0.77}$ printed in a box above the corresponding node in part 8(a) represents the value of similarity. Since there are not sub-holons to ask for their proposals, super-holon *3:X* is created and both *1:X* and the newly created holons for that algorithm *4:X* join that super-holon (part 8(d)), and the capabilities of the holons are updated, as explained in Algorithm 2. The remaining parts of the figure show the same set of steps to handle three more new incoming algorithm info queries, and hence, for the sake of space, we do not explain them further.

C.2 Address Update Example

Figure 9 demonstrates the way addresses are updated during the training phase. In part 9(a), the target address entries of holon **0**'s memory for queries q_1 and q_2 are pointing to atomic holon **1**. In part 9(b), the new holon **3** is created and inserted because of query q_3. As a result, the memory entries of holon **0** are updated to point to the newly created super-holon **2**, and holon **2** per se,

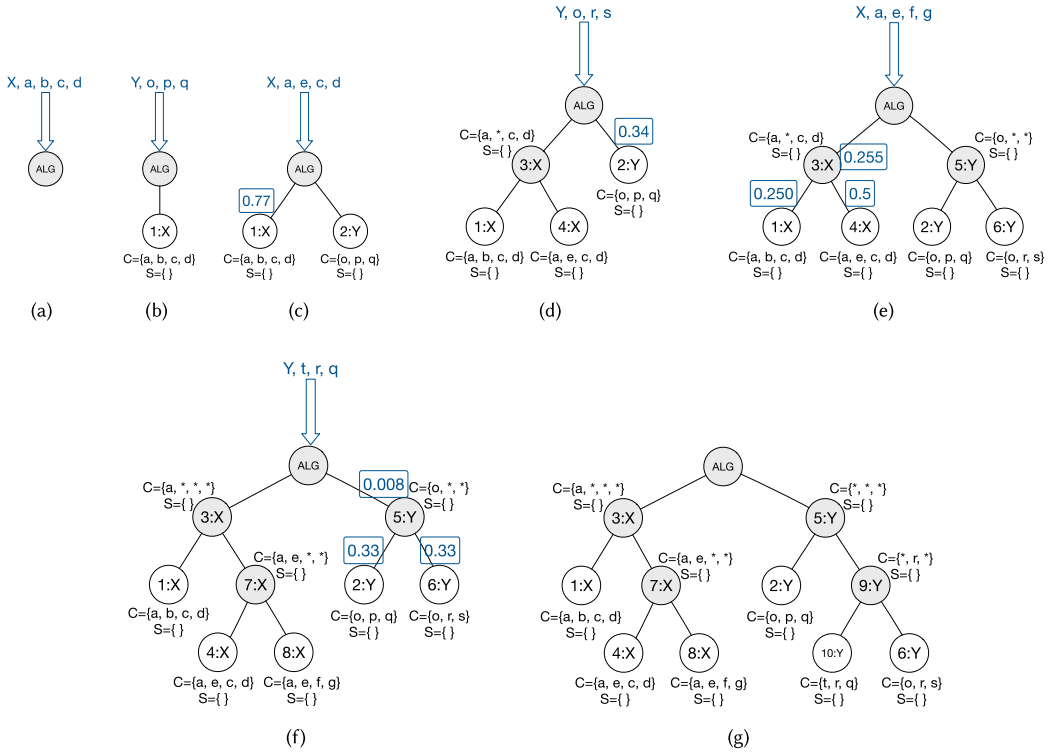


Fig. 8. The step-by-step demonstration of Algorithms 1 and 2 in an example.

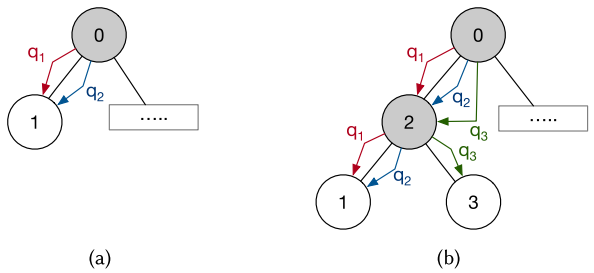


Fig. 9. An example demonstrating the way the access addresses are updated when a new holon is created.

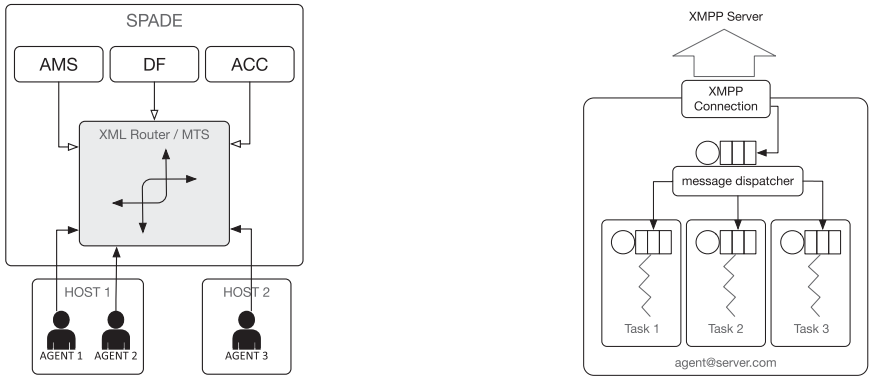
points to the holon **1** now. As soon as the first pass of q_3 finishes, the corresponding addresses (shown in green color) are created for this query as explained before.

D EXPERIMENT SETTINGS

This section provides more details about the implementation of HAMLET and the configurations we used to assess its functionality and flexibility in real-world settings.

D.1 Implementation

D.1.1 SPADE. SPADE is fully FIPA-compliant and supports asynchronous running of agents together with the inter-agent communications based on the **Extensible Messaging and Presence**



(a) The platform model (b) The agent model

Fig. 10. The architecture of the SPADE agent development framework [33].

Protocol (XMPP). Moreover, it provides a set of helpful features that facilitate the deployment of our platform on a network of computers. Some of such characteristics are [33]:

- the flexibility in inter-operating with other agent development platforms, thanks to its FIPA compliance;
- **multi-user Conference (MUC)** that provides the capability to create forums of agents;
- providing multiple built-in behavior models, such as cyclic, recurring, one-shot, timeout, and event-based finite state machines; and
- featuring customized agent presence notification and P2P agent communication capability.

SPADE's platform and agent models are depicted in Figure 10. As it can be seen, its main platform is outlined based on the multi-agent architecture standards recommended by FIPA [28]. To put it concisely, the platform (Figure 10(a)) is composed of four components: the **Agent Management System (AMS)** to supervise SPADE, the **Directory Facilitator (DF)** to provide information about the agents and their services, the **Agent Communication Channel (ACC)** to manage the communications between the agents and system components, and the XML router as the **Message Transport System (MTS)**. However, the agent model (Figure 10(b)) comprises tasks, as the executable processes of the agent; and the message dispatcher that collects the arrived messages and redirects them to the appropriate task queues.

D.1.2 SPADE-based Implementation of HAMLET. To develop the holonic structure, we specialized SPADE's agent model by adding the internal components, summarized in Figure 3, to the built-in elements such as the message dispatcher. Figure 11 illustrates a high-level view of the implemented classes and their relationships with each other. The shaded area in this diagram particularly highlights the components that HAMLET implements based on its architecture. Class *Holon*, as an abstract class, defines the basic data structures, properties, and behaviors common in all holon types. The children of this class try to customize the provided basic architecture and interfaces with more specific and task-oriented components. Furthermore, separating different holon types in different classes helps us effectively enforce the restrictions, such the multiplicity and holarchical relationships, defined in HAMLET. The other classes presented in Figure 11, namely, *PRS*, *VIZ*, and *User*, are technically the SPADE agents with particular functionalities. It is worth noting that due to the flexible structure of HAMLET, the functionality and services of the test bed can be easily extended by adding more agents and properly connecting them to the other holons.

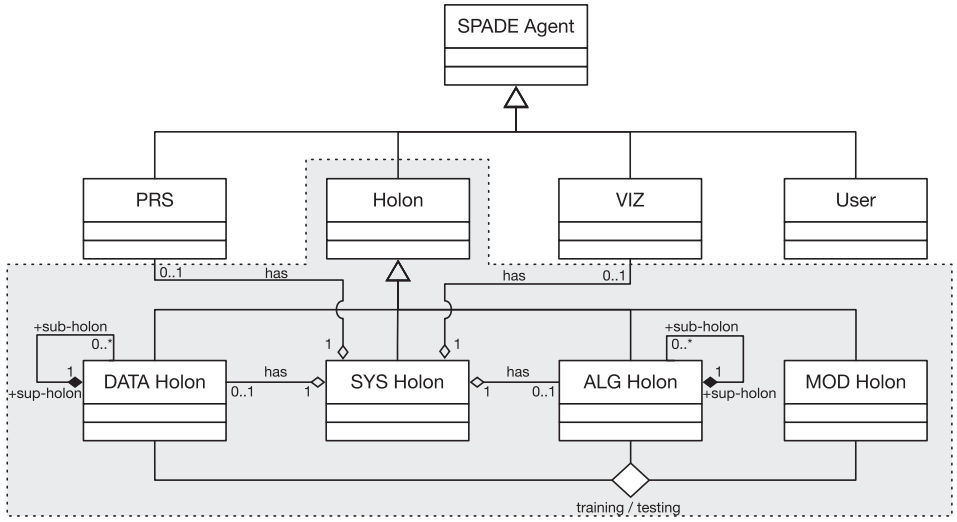


Fig. 11. The classes diagram of the implemented HAMLET elements and their relationship.

D.2 Used Machine Learning Datasets and Algorithms

Tables 8 and 9 summarize the ML resources that are utilized for empirical analysis. In Table 8, for each algorithm, the columns *parameters* and *id,* respectively, hold the list of the used parameters and the identifier that is used for the presentation purposes. Please note that, for the sake of space limitation and clarity, we have only listed the parameters that have different values from the default ones and we have listed the complete list of the parameters for each algorithm as the note in Table 8. Additionally, Table 9 lists the datasets that are used for each machine learning task. For classification and regression, we have divided each dataset into training and testing sets consisting of 60% and 40% of the original instances, respectively. For the clustering task, however, we have utilized 100% of the data.

D.3 Experimental Diagrams

D.3.1 The Holarchical Structure. Figure 12 depicts the multi-level layout of the holarchy generated during the training phase. The figures on the left-hand side column represent the structure of the holarchy without considering the model holons, and the ones on the right-hand side focus solely on the interconnection between the atomic algorithm/data holons and the created models at the lowest level of the holarchy. These two views are provided mainly for the sake of clarity. Furthermore, the sub-figures at each row pertain to the holarchy after a particular task, i.e., sub-figures 12(a) and 12(b) are for after training the classification algorithms, sub-figures 12(c) and 12(d) are for after training the regression algorithms, sub-figures 12(e) and 12(f) are for after running the clustering algorithms, and finally, sub-figures 12(g) and 12(h) are for after adding test data. There are a few additional points about the figures worth noting. First, in Figure 12(d), the structure is composed of two separate communities, which is because of the fact that the classification and regression algorithms do not share any dataset during the training phase. Second, in Figure 12(h), there are some DataH holons in the vicinity of communities that are not connected to any other holons. These are the test datasets that are added to the holarchy and because they are not employed in any training task, they do not hold any connections to the model holons. And finally, the nodes of the structures are positioned automatically by the visualization algorithm, therefore, the

Table 8. The Details of the Used Algorithms

classification			*regression*			*clustering*		
id	*name*	*parameters*	*id*	*name*	*parameters*	*id*	*name*	*parameters*
A01	SVC[1]	kernel=linear	**A09**	Linear[6]	defaults	**A17**	KMeans[12]	defaults*
A02	SVC	kernel=sigmoid	**A10**	Ridge[7]	fit_inercept=False	**A18**	KMeans	algorithm=full*
A03	SVC	$\gamma = 0.001$	**A11**	Ridge	$\alpha = 0.5$	**A19**	MBKMeans[13]	defaults*
A04	SVC	$C = 100, \gamma = 0.001$	**A12**	KRR[8]	defaults	**A20**	DBSCAN[14]	defaults
A05	NuSVC[2]	defaults	**A13**	Lasso[9]	$\alpha = 0.1$	**A21**	DBSCAN	metric=cityblock
A06	ComNB[3]	defaults	**A14**	NuSVR[10]	defaults	**A22**	DBSCAN	metric=cosine
A07	DTree[4]	defaults	**A15**	NuSVR	$\nu = 0.1$	**A23**	Birch[15]	defaults
A08	NrCent[5]	defaults	**A16**	ElasNet[11]	defaults	**A24**	HAC[16]	defaults*

[1]**C-Support Vector Classification** [14]. Defaults:(C=1.0, kernel="rbf," degree=3, γ="scale," coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape="ovr," break_ties=False) [1].
[2]**Nu-Support Vector Classification** [14]. Defaults:(nu=0.5, kernel="rbf," degree=3, γ="scale," coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape="ovr," break_ties=False) [1].
[3]**Complement Naive Bayes classifier** [56]. Defaults:(α=1.0, fit_prior=True, class_prior=None, norm=False) [1].
[4]**Decision Tree Classifier** [35]. Defaults:(riterion="gini," splitter="best," max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort="deprecated," ccp_alpha=0.0) [1].
[5]**Nearest Centroid Classifier** [65]. Defaults:(metric="euclidean," shrink_threshold=None) [1].
[6]**Ordinary Least Squares Linear Regression**. Defaults:(fit_intercept=True, normalize=False, copy_X=True, n_jobs=None) [1].
[7]**Ridge Regression** [38]. Defaults:($\alpha = 1.0$, fit_intercept=True, normalize=False, copy_X=True, max_iter=None, tol=0.001, solver=auto) [1].
[8]**Kernel Ridge Regression** [50]. Defaults: (α=1, kernel="linear," γ=None, degree=3, coef0=1, kernel_params=None) [1].
[9]**least Absolute Shrinkage and Selection Operator** [64]. Defaults:(α=1.0, fit_intercept=True, normalize=False, precompute=False, copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selection="cyclic") [1].
[10]**Nu Support Vector Regression** [14]. Defaults:(ν=0.5, C=1.0, kernel="rbf," degree=3, γ="scale," coef0=0.0, shrinking=True, tol=0.001, cache_size=200, verbose=False, max_iter=-1) [1].
[11]**Elastic Net Regression** [76]. Defaults:(α=1.0, l1_ratio=0.5, fit_intercept=True, normalize=False, precompute=False, max_iter=1000, copy_X=True, tol=0.0001, warm_start=False, positive=False, random_state=None, selection="cyclic") [1].
[12]**K-Means Clustering** [47]. Defaults:(n_clusters=8, init="k-means++," n_init=10, max_iter=300, tol=0.0001, precompute_distances="deprecated," verbose=0, random_state=None, copy_x=True, n_jobs="deprecated," algorithm="auto") [1].
[13]**Mini-Batch K-Means Clustering** [61]. Defaults:(n_clusters=8, init="k-means++," max_iter=100, batch_size=100, verbose=0, compute_labels=True, random_state=None, tol=0.0, max_no_improvement=10, init_size=None, n_init=3, reassignment_ratio=0.01) [1].
[14]**Density-Based Spatial Clustering of Applications with Noise** [22]. Defaults:(ϵ=0.5, min_samples=5, metric="euclidean," metric_params=None, algorithm="auto," leaf_size=30, p=None, n_jobs=None) [1].
[15] **Birch Clustering** [75]. Defaults:(threshold=0.5, branching_factor=50, n_clusters=3, compute_labels=True, copy=True) [1].
[16]**Hierarchical Agglomerative Clustering** [58]. Defaults:(n_clusters=2, affinity="euclidean," memory=None, connectivity=None, compute_full_tree="auto," linkage="ward," distance_threshold=None) [1].
*The number of the clusters is set equal to the number of true classes.

Table 9. Details of the Used Datasets

name	*classes/targets*	*samples per class*	*total samples*	*dimensionality*	*features*
classification:					
Iris [27]*	3	[50, 50, 50]	150	4	real, positive
Wine [45]*	3	[59, 71, 48]	178	13	real, positive
Breast cancer [71]	2	[212, 358]	569	30	real, positive
Digits [4]*, †	10	about 180	1,797	64	integers [0, 16]
Art. Class., [1]	3	[300, 300, 300]	900	20	real (−7.3, 8.9)**
Art. Moon, [2]	2	[250, 250]	500	2	real (−1.2, 2.2)**
regression:					
Boston [34]	real [5, 50]	–	506	13	real, positive
Diabetes [18]	integer [25, 346]	–	442	10	real (−0.2, 0.2)
Art. Regr.[3]	real (−488.1, 533.2)	–	200	20	real (−4, 4)

*Also used in clustering experiments.
**To prevent the problem that some algorithms have with negative features, the values are normalized into [0,1].
†This is a copy of the test set of the UCI ML hand-written digits datasets.
[1]Artificially made using `make_classification` function of scikit-learn library [1].
[2]Artificially made using `make_moons` function of scikit-learn library [1].
[3]Artificially made using `make_regression` function of scikit-learn library [1].

algorithm sub-structures in Figures 12(e) and 12(g) are exactly the same, though they are drawn differently.

D.3.2 Plots generated by the Visualizing Agent. The style used to report the results—a separate plot for each performance measure in this example—is decided by the auxiliary visualization agent, VIZ, and is not enforced by HAMLET.

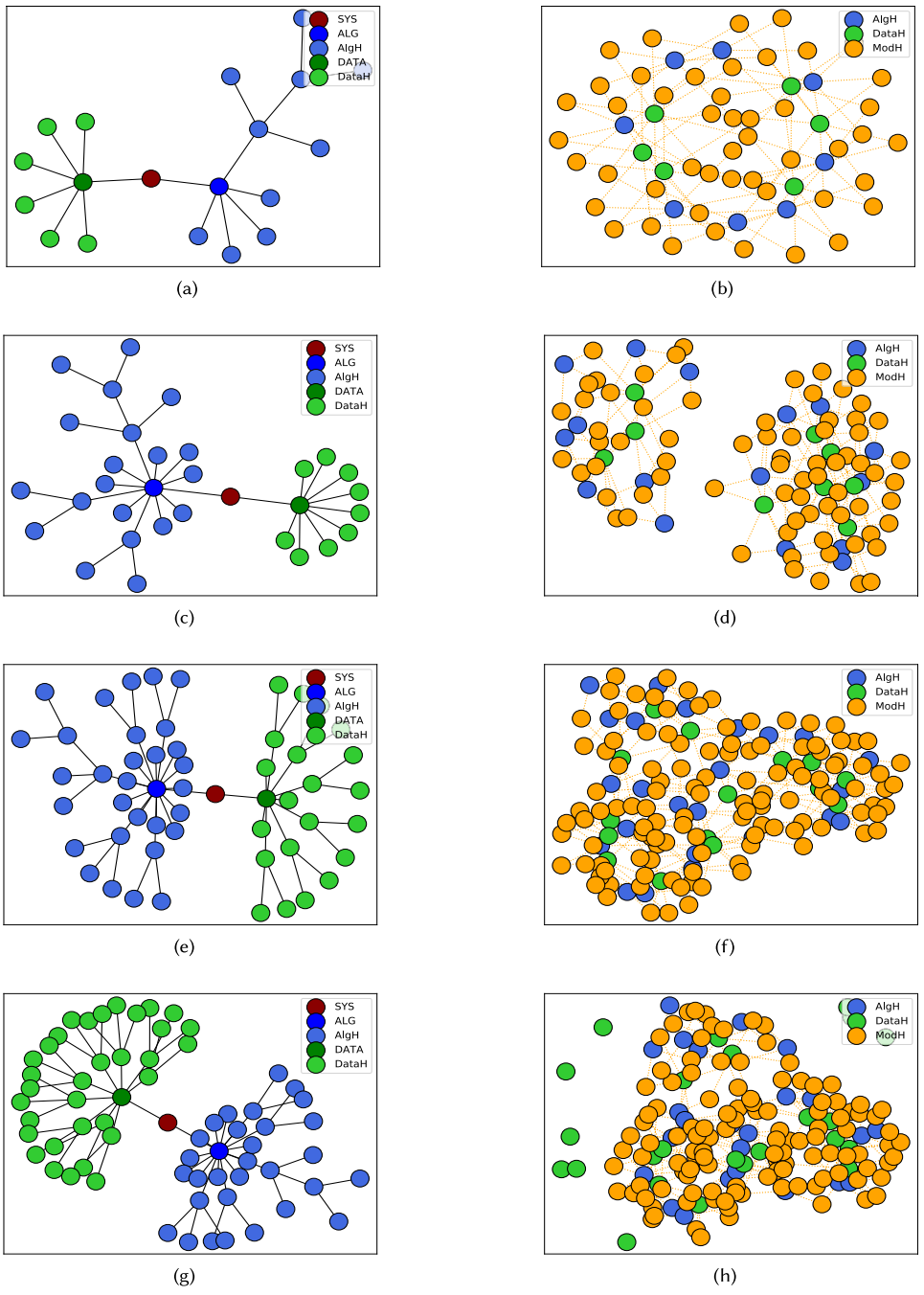


Fig. 12. The holarchical (left) and the atomic-level structures (right) after classification (12(a), 12(b)), regression (12(c), 12(d)), clustering (12(e), 12(f)), and adding test datasets (12(f), 12(h)).

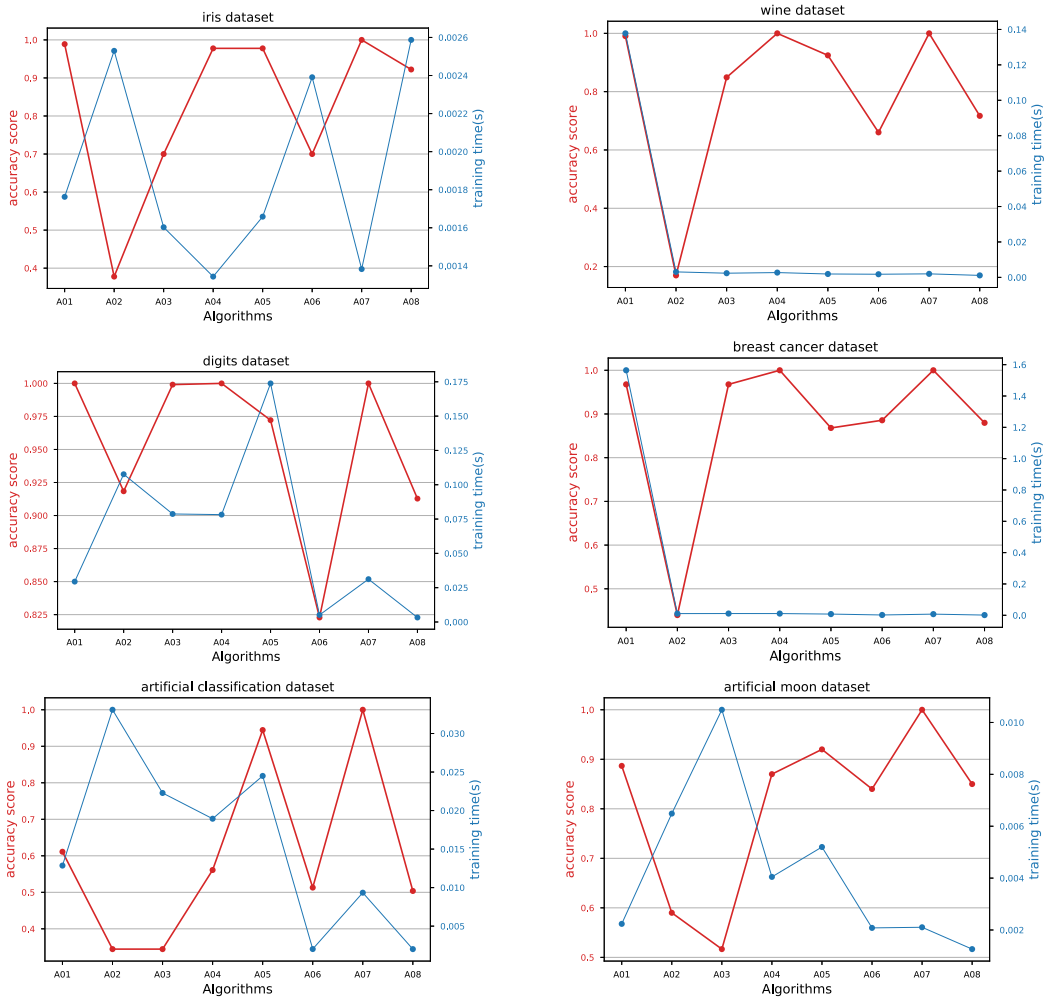


Fig. 13. Accuracy and time of training each classification algorithm on a specific dataset.

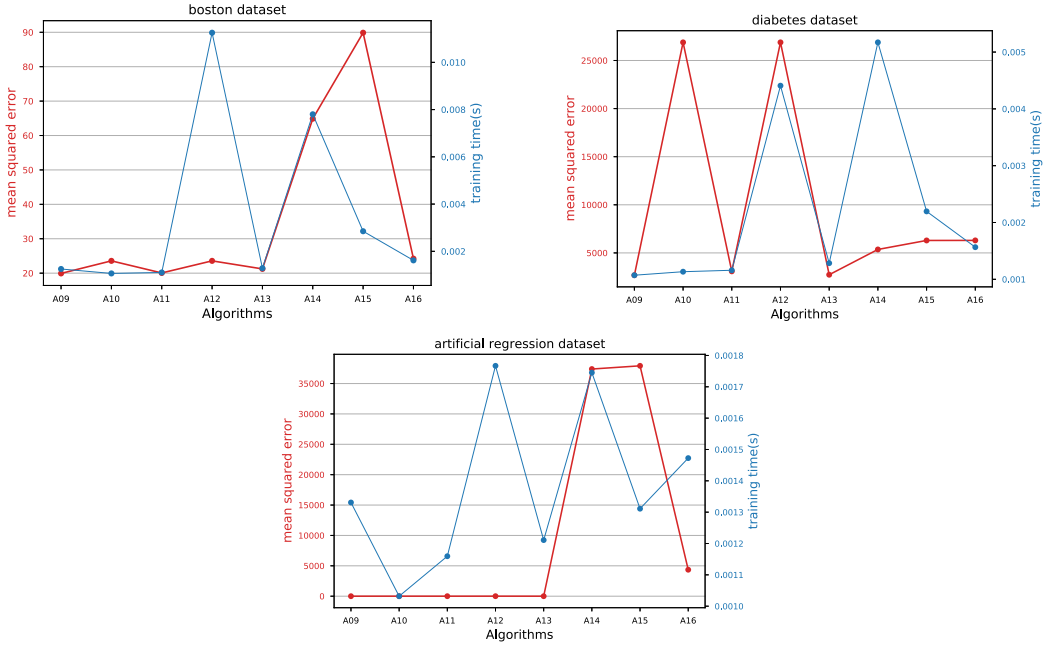


Fig. 14. Mean squared error and time of training each regression algorithm on a specific dataset.

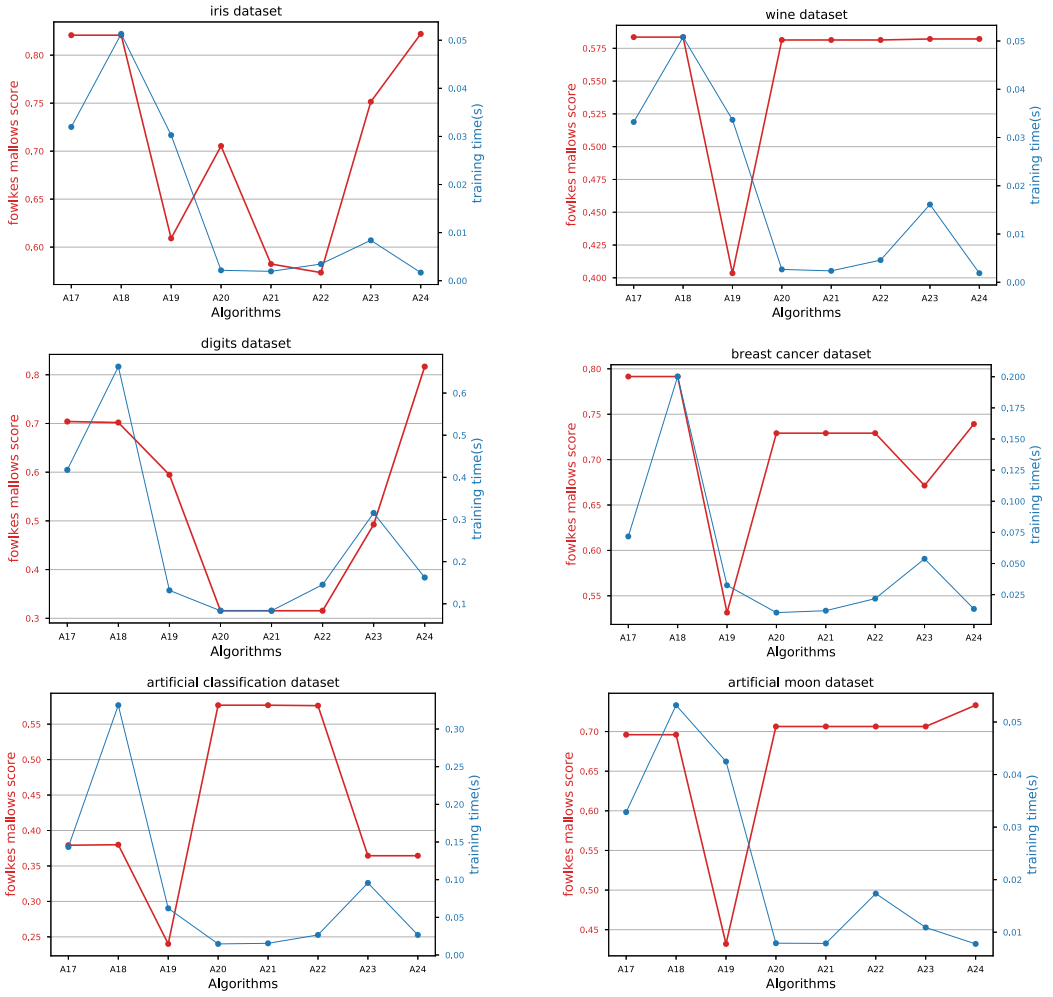


Fig. 15. Fowlkes-Mallows score and time of running each clustering algorithm on a specific dataset.

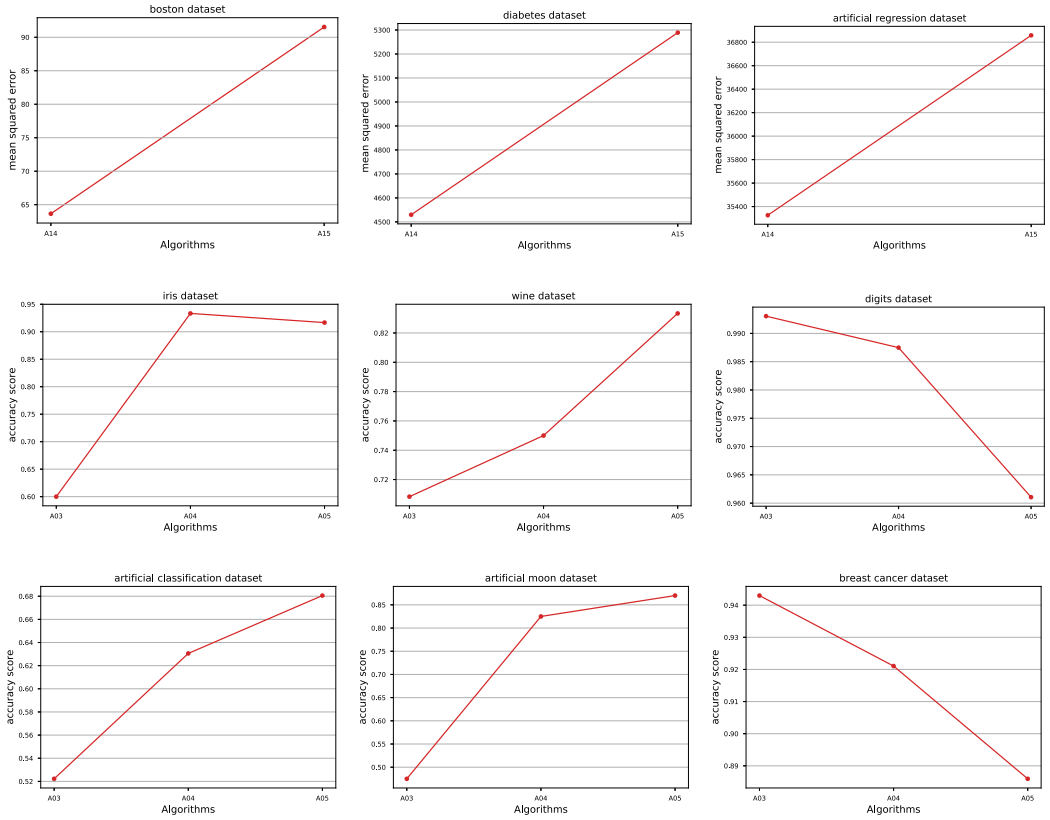


Fig. 16. The results of testing all classification and regression algorithms that have parameter *kernel* equal to *rbf* on all proper datasets, in terms of accuracy and mean squared error.

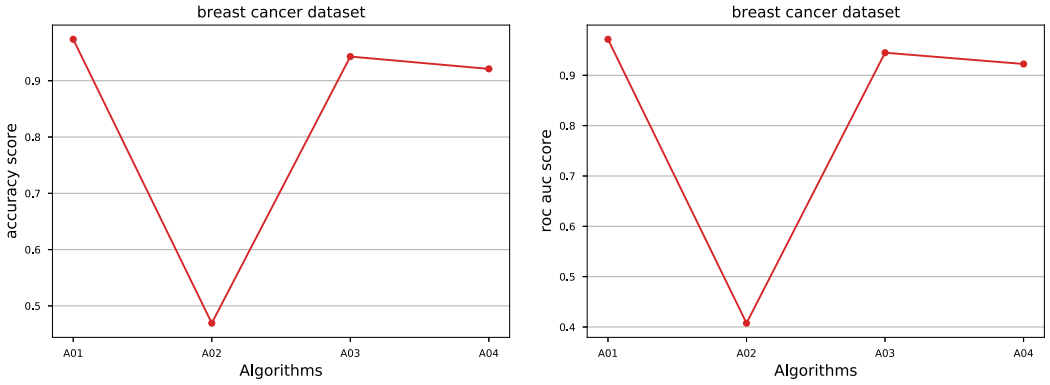


Fig. 17. The results of testing all SVC algorithms with any parameter on the breast cancer dataset, in terms of accuracy and area under the ROC curve score.

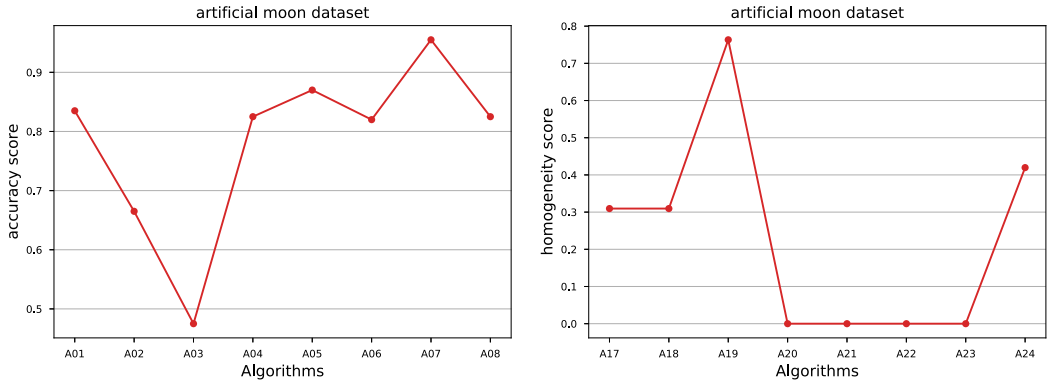


Fig. 18. The results of testing all algorithms with any parameter on the artificial moon dataset, in terms of accuracy, mean square error, and/or clustering homogeneity score.

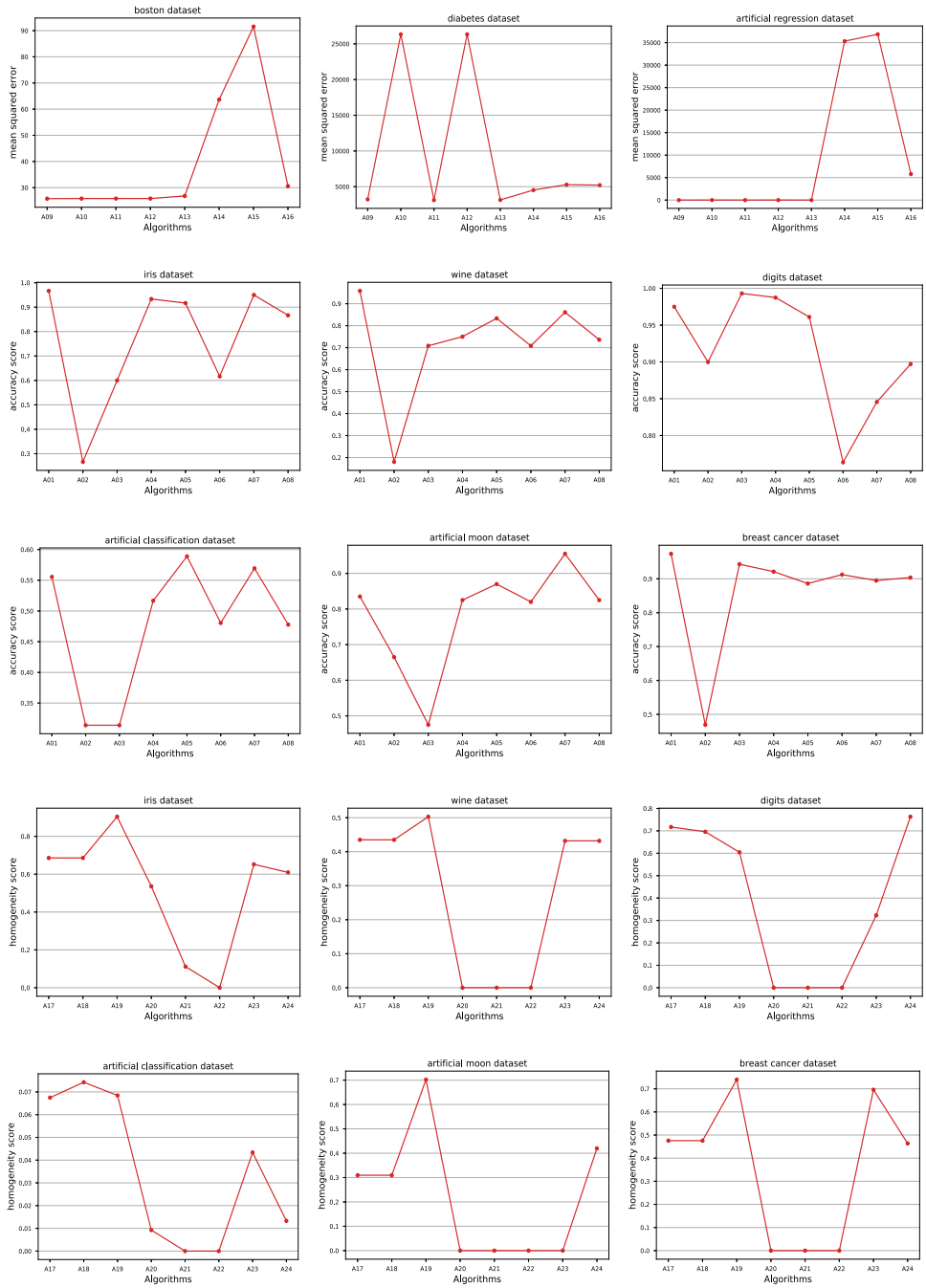


Fig. 19. The results of testing all algorithms with any parameter on all the test datasets, in terms of accuracy, mean square error, and/or clustering homogeneity score.

REFERENCES

[1] Scikit-learn API Reference. (n.d.). Retrieved April 1, 2020, from https://scikit-learn.org/stable/modules/classes.html/.

[2] Emmanuel Adam, René Mandiau, and Christophe Kolski. 2000. HOMASCOW: A holonic multi-agent system for cooperative work. In *Proceedings of the 11th International Workshop on Database and Expert Systems Applications*. IEEE, 247–253.

[3] Kamal Ali Albashiri, Frans Coenen, and Paul Leng. 2008. EMADS: An extendible multi-agent data miner. In *Proceedings of the International Conference on Innovative Techniques and Applications of Artificial Intelligence*. Springer, 263–275.

[4] Ethem Alpaydin and Cenk Kaynak. 1998. Cascading classifiers. *Kybernetika* 34, 4 (1998), 369–374.

[5] Estefanía Argente, Vicent Botti, and Vicente Julian. 2009. GORMAS: An organizational-oriented methodological guideline for open MAS. In *Proceedings of the International Workshop on Agent-oriented Software Engineering*. Springer, 32–47.

[6] Susan Athey. 2018. The impact of machine learning on economics. In *The Economics of Artificial Intelligence: An Agenda*. University of Chicago Press, 507–547.

[7] Stuart Bailey, Robert Grossman, Harimath Sivakumar, and A. Turinsky. 1999. Papyrus: A system for data mining over local and wide area clusters and super-clusters. In *Proceedings of the ACM/IEEE Conference on Supercomputing*. 63–es.

[8] Bernd Bischl, Michel Lang, Lars Kotthoff, Julia Schiffner, Jakob Richter, Erich Studerus, Giuseppe Casalicchio, and Zachary M. Jones. 2016. MLR: Machine learning in R. *J. Mach. Learn. Res.* 17, 1 (2016), 5938–5942.

[9] Ekaba Bisong. 2019. Google BigQuery. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Springer, 485–517.

[10] Stefan Bosse. 2017. Incremental distributed learning with JavaScript agents for earthquake and disaster monitoring. *Int. J. Distrib. Syst. Technol.* 8, 4 (2017), 34–53.

[11] Hans Jurgen Burckert, Klaus Fischer, and Gero Vierke. 1998. Transportation scheduling with holonic MAS: The TeleTruck approach. In *Proceedings of the International Conference on the Practical Application of Intelligent Agents and Multi-agent Technology*.

[12] Longbing Cao, Vladimir Gorodetsky, and Pericles A. Mitkas. 2009. Agent mining: The synergy of agents and data mining. *IEEE Intell. Syst.* 24, 3 (2009), 64–72.

[13] Santhana Chaimontree, Katie Atkinson, and Frans Coenen. 2012. A framework for multi-agent based clustering. *Auton. Agents Multi-agent Syst.* 25, 3 (2012), 425–446.

[14] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.* 2, 3 (2011), 1–27.

[15] Călin Ciufudean and Constantin Filote. 2011. Artificial social models for holonic systems. In *Proceedings of the International Conference on Industrial Applications of Holonic and Multi-agent Systems*. Springer, 133–142.

[16] Massimo Cossentino, Nicolas Gaud, Vincent Hilaire, Stéphane Galland, and Abderrafiâa Koukam. 2010. ASPECS: An agent-oriented software process for engineering complex systems. *Auton. Agents Multi-agent Syst.* 20, 2 (2010), 260–304.

[17] Ireneusz Czarnowski and Piotr Jędrzejowicz. 2013. Machine learning and multiagent systems as interrelated technologies. In *Agent-based Optimization*. Springer, 1–28.

[18] Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. 2004. Least angle regression. *Ann. Statist.* 32, 2 (2004), 407–499.

[19] Ahmad Esmaeili, Nasser Mozayani, Mohammad Reza Jahed-Motlagh, and Eric T. Matson. 2019. Towards topological analysis of networked holonic multi-agent systems. In *Proceedings of the International Conference on Practical Applications of Agents and Multi-agent Systems*. Springer, 42–54.

[20] Ahmad Esmaeili, Nasser Mozayani, Mohammad Reza Jahed Motlagh, and Eric T. Matson. 2016. The impact of diversity on performance of holonic multi-agent systems. *Eng. Applic. Artif. Intell.* 55 (2016), 186–201.

[21] Ahmad Esmaeili, Nasser Mozayani, Mohammad Reza Jahed Motlagh, and Eric T. Matson. 2017. A socially-based distributed self-organizing algorithm for holonic multi-agent systems: Case study in a task environment. *Cogn. Syst. Res.* 43 (2017), 21–44.

[22] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Conference on Knowledge Discovery and Data Mining*. 226–231.

[23] Tom Fawcett. 2006. An introduction to ROC analysis. *Patt. Recogn. Lett.* 27, 8 (2006), 861–874.

[24] Solomia Fedushko and Taras Ustyianovych. 2019. Predicting pupil's successfulness factors using machine learning algorithms and mathematical modelling methods. In *Proceedings of the International Conference on Computer Science, Engineering and Education Applications*. Springer, 625–636.

[25] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*. 2962–2970.

[26] Klaus Fischer, Michael Schillo, and Jörg Siekmann. 2003. Holonic multiagent systems: A foundation for the organisation of multiagent systems. In *Proceedings of the International Conference on Industrial Applications of Holonic and Multi-agent Systems*. Springer, 71–80.
[27] Ronald A. Fisher. 1936. The use of multiple measurements in taxonomic problems. *Ann. Eugen.* 7, 2 (1936), 179–188.
[28] Foundation for Intelligent Physical Agents. 2002. *FIPA Abstract Architecture Specification*. Technical Report SC00001L. Foundation for Intelligent Physical Agents.
[29] Edward B. Fowlkes and Colin L. Mallows. 1983. A method for comparing two hierarchical clusterings. *J. Amer. Statist. Assoc.* 78, 383 (1983), 553–569.
[30] Sannasi Ganapathy, P. Yogesh, and Arputharaj Kannan. 2012. Intelligent agent-based intrusion detection system using enhanced multiclass SVM. *Computat. Intell. Neurosci.* 2012, 9 (2012). https://dl.acm.org/doi/abs/10.1155/2012/850259.
[31] Sunyue Geng, Sifeng Liu, Zhigeng Fang, and Su Gao. 2021. An agent-based clustering framework for reliable satellite networks. *Reliab. Eng. Syst. Safety* 212, Article C (2021), 107630. https://ideas.repec.org/a/eee/reensy/v212y2021ics095183202100171x.html.
[32] Vladimir Gorodetsky, Oleg Karsaeyv, and Vladimir Samoilov. 2003. Multi-agent technology for distributed data mining and classification. In *Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology*. IEEE, 438–441.
[33] Miguel Escrivá Gregori, Javier Palanca Cámara, and Gustavo Aranda Bada. 2006. A jabber-based multi-agent system platform. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*. 1282–1284.
[34] David Harrison Jr and Daniel L. Rubinfeld. 1978. Hedonic housing prices and the demand for clean air. (1978). https://www.sciencedirect.com/science/article/abs/pii/0095069678900062.
[35] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science & Business Media.
[36] Vincent Hilaire, Abder Koukam, Pablo Gruer, and Jean-Pierre Müller. 2000. Formal specification and prototyping of multi-agent systems. In *Proceedings of the International Workshop on Engineering Societies in the Agents World*. Springer, 114–127.
[37] Vincent Hilaire, Abder Koukam, and Sebastian Rodriguez. 2008. An adaptative agent architecture for holonic multiagent systems. *ACM Trans. Auton. Adapt. Syst.* 3, 1 (2008), 1–24.
[38] Arthur E. Hoerl and Robert W. Kennard. 1970. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics* 12, 1 (1970), 55–67.
[39] Markus Hofmann and Ralf Klinkenberg. 2016. *RapidMiner: Data Mining Use Cases and Business Analytics Applications*. CRC Press.
[40] Hillol Kargupta, B. Park, Daryl Hershberger, and Erik Johnson. 1999. Collective data mining: A new perspective toward distributed data mining. *Adv. Distrib. Parallel Knowl. Discov.* 2 (1999), 131–174.
[41] Hillol Kargupta, Brian Stafford, and Ilker Hamzaoglu. 1997. *Web Based Parallel/Distributed Medical Data Mining Using Software Agents*. Technical Report. Los Alamos National Lab., NM.
[42] H. Kergupta, I. Hamzaoglu, and B. Stafford. 1997. Scalable, distributed data mining using an agent based architecture (PADMA). In *Proc. High Performance Computing*, Vol. 97. AAAI Press.
[43] Arthur Koestler. 1968. The ghost in the machine. Macmillan.
[44] Konstantinos G. Liakos, Patrizia Busato, Dimitrios Moshou, Simon Pearson, and Dionysis Bochtis. 2018. Machine learning in agriculture: A review. *Sensors* 18, 8 (2018), 2674.
[45] Kevin Bache and Moshe Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.
[46] Bin Liu, Shu Gui Cao, and Wu He. 2011. Distributed data mining for e-business. *Inf. Technol. Manag.* 12, 2 (2011), 67–79.
[47] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE Trans. Inf. Theor.* 28, 2 (1982), 129–137.
[48] Gianfranco Lombardo, Paolo Fornacciari, Monica Mordonini, Michele Tomaiuolo, and Agostino Poggi. 2019. A multi-agent architecture for data analysis. *Fut. Internet* 11, 2 (2019), 49.
[49] Francisco Maturana, Weiming Shen, and Douglas H. Norrie. 1999. MetaMorph: An adaptive agent-based architecture for intelligent manufacturing. *Int. J. Product. Res.* 37, 10 (1999), 2159–2173.
[50] Kevin P. Murphy. 2012. *Machine Learning: A Probabilistic Perspective*. The MIT Press.
[51] James Odell, Marian Nodine, and Renato Levy. 2004. A metamodel for agents, roles, and groups. In *Proceedings of the International Workshop on Agent-oriented Software Engineering*. Springer, 78–92.
[52] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* 12 (2011), 2825–2830.
[53] Shengquan Peng, Snehasis Mukhopadhyay, R. Raje, and Mathew Palakal. 2001. A comparison between single-agent and multi-agent classification of documents. In *Proceedings of the 10th Heterogeneous Computing Workshop*. 20090–2.

[54] Rajeev R. Raje, Snehasis Mukhopadhyay, Michael Boyles, Nila Patel, Artur Papiez, and J. Mostafa. 1998. On designing and implementing a collaborative system using the distributed-object model of Java-RMI. *Parallel Distrib. Comput. Pract. J.* 1, 4 (1998), 3–14.

[55] Alvin Rajkomar, Jeffrey Dean, and Isaac Kohane. 2019. Machine learning in medicine. *New Eng. J. Med.* 380, 14 (2019), 1347–1358.

[56] Jason D. Rennie, Lawrence Shih, Jaime Teevan, and David R. Karger. 2003. Tackling the poor assumptions of naive bayes text classifiers. In *Proceedings of the 20th International Conference on Machine Learning (ICML'03)*. 616–623.

[57] Sebastian Rodriguez, Vincent Hilaire, and Abder Koukam. 2003. Towards a methodological framework for holonic multi-agent systems. In *Proceedings of the 4th International Workshop of Engineering Societies in the Agents World*. 29–31.

[58] Lior Rokach and Oded Maimon. 2005. Clustering methods. In *Data Mining and Knowledge Discovery Handbook*. Springer, 321–352.

[59] Andrew Rosenberg and Julia Hirschberg. 2007. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. 410–420.

[60] Dominik Ryzko. 2020. *Modern Big Data Architectures: A Multi-agent Systems Perspective*. John Wiley & Sons.

[61] David Sculley. 2010. Web-scale k-means clustering. In *Proceedings of the 19th International Conference on World Wide Web*. 1177–1178.

[62] Smith. 1980. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. Comput.* C-29, 12 (1980), 1104–1113.

[63] Salvatore J. Stolfo, Andreas L. Prodromidis, Shelley Tselepis, Wenke Lee, Dave W. Fan, and Philip K. Chan. 1997. JAM: Java agents for meta-learning over distributed databases. In *Proceedings of the Conference on Knowledge Discovery and Data Mining*, Vol. 97. 74–81.

[64] Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *J. Roy. Statist. Societ.: Series B (Methodol.)* 58, 1 (1996), 267–288.

[65] Robert Tibshirani, Trevor Hastie, Balasubramanian Narasimhan, and Gilbert Chu. 2002. Diagnosis of multiple cancer types by shrunken centroids of gene expression. *Proc. Nat. Acad. Sci.* 99, 10 (2002), 6567–6572.

[66] Youssef Tounsi, Houda Anoun, and Larbi Hassouni. 2020. CSMAS: Improving multi-agent credit scoring system by integrating big data and the new generation of gradient boosting algorithms. In *Proceedings of the 3rd International Conference on Networking, Information Systems & Security*. 1–7.

[67] Jan Tozicka, Michael Rovatsos, and Michal Pechoucek. 2007. A framework for agent-based distributed machine learning and data mining. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*. 1–8.

[68] Jessica Vamathevan, Dominic Clark, Paul Czodrowski, Ian Dunham, Edgardo Ferran, George Lee, Bin Li, Anant Madabhushi, Parantu Shah, Michaela Spitzer, et al. 2019. Applications of machine learning in drug discovery and development. *Nat. Rev. Drug Discov.* 18, 6 (2019), 463–477.

[69] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. 2013. OpenML: Networked science in machine learning. *SIGKDD Explor.* 15, 2 (2013), 49–60.

[70] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S. Rellermeyer. 2020. A survey on distributed machine learning. *ACM Comput. Surv.* 53, 2 (2020), 1–33.

[71] William H. Wolberg, W. Nick Street, and Olvi L. Mangasarian. 1994. Machine learning techniques to diagnose breast cancer from image-processed nuclear features of fine needle aspirates. *Cancer Lett.* 77, 2-3 (1994), 163–171.

[72] Michael Wooldridge. 2009. *An Introduction to Multiagent Systems*. John Wiley & Sons.

[73] Bang Xiang Yong and Alexandra Brintrup. 2019. Multi agent system for machine learning under uncertainty in cyber physical manufacturing system. In *Proceedings of the International Workshop on Service Orientation in Holonic and Multi-agent Manufacturing*. Springer, 244–257.

[74] Chengqi Zhang, Zili Zhang, and Longbing Cao. 2005. Agents and data mining: Mutual enhancement by integration. In *Proceedings of the International Workshop on Autonomous Intelligent Systems: Agents and Data Mining*. Springer, 50–61.

[75] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. BIRCH: An efficient data clustering method for very large databases. *ACM SIGMOD Rec.* 25, 2 (1996), 103–114.

[76] Hui Zou and Trevor Hastie. 2005. Regularization and variable selection via the elastic net. *J. Roy. Statist. Societ.: Series B (Methodol.)* 67, 2 (2005), 301–320.

Received October 2020; revised March 2022; accepted April 2022